National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canadä

UNIVERSITY OF ALBERTA

A VMEbus CONTROLLER FOR THE PUMA 560 ROBOT

BY

Lee Edmund Kruszewski

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE
IN
CONTROL SYSTEMS

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA
FALL 1990

National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Service des thèses canadiennes

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-65066-4

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR:	LEE EDMUND KRUSZEWSKI
TITLE OF THESIS:	A VMEbus CONTROLLER FOR THE PUMA 560 ROBOT
DEGREE:	MASTER OF SCIENCE
YEAR THIS DEGREE GRANTED:	FALL, 1990

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Student's Signature)

Student's Permanent Address:

#301, 10645, 79 Avenue
EDMONTON, ALBERTA,
T6E 1E5

Date: ______________

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled A VMEbus CONTROLLER FOR THE PUMA 560 ROBOT submitted by LEE EDMUND KRUSZEWSKI in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN ELECTRICAL ENGINEERING in CONTROL SYSTEMS.

V.G. Gourishankar
(Supervisor)

R. E. Rink

R.E. Rink

Hong Zhang

Hong Zhang

Date: August 8, 1990

ABSTRACT

A customized, VMEbus based controller for the Unimate PUMA 560 robot is described. Current robot control systems are reviewed to place this controller in context. Hardware and software, designed to supplant the Unimate controller at the servo level, is described. Experimental results are presented to demonstrate the operation of the new controller. Conclusions are drawn and future research directions are recommended.

ACKNOWLEDGEMENTS

I would like to thank NSERC and the Department of Electrical Engineering, University of Alberta for their financial support. I would also like to thank my supervisor, Dr. V.G. Gourishankar, for his guidance in preparing this thesis.

TABLE OF CONTENTS

1. **INTRODUCTION** 1

2. **LITERATURE REVIEW** 7

2.1 Approaches and Concepts Used in Robot Controller Design 7

2.2 An Overview of Robot Control Systems 11

2.2.1 Tabular Comparison 12

2.2.2 Comparison of System Architectures . 17

2.2.2.1 The CHIMERA Standard 17

2.2.2.2 Other Systems 19

2.2.3 Robot Controllers for the PUMA 560 . 24

2.3 Comparison of Software Environments 28

3. **DESCRIPTION OF PROPOSED CONTROLLER HARDWARE** . . . 33

3.1 Design Goals and Methodology 33

3.2 System Configuration 35

3.3 Description of Individual Boards 41

3.3.1 System Design Decisions and Goals . . 41

3.3.2 Commercial Boards 42

3.3.3 AMline Decoder Board (ADB) 43

3.3.4 VMEbus Encoder Board (VEB) 45

3.3.4.1 VEB Introduction 45

3.3.4.2 VMEbus Interface Design . . . 47

3.3.4.3 Quadrature Decoder Circuitry Design 49
3.3.4.4 System Testing and Integration 52
3.3.5 PUMA Interface Board (PIB) 53
3.3.6 Current Measurement Amplifier/Sensor Board 57

4. **DESCRIPTION OF PROPOSED CONTROLLER SOFTWARE** . . . 59
4.1 The Software Environment 59
4.2 Description of Software 63
4.2.1 Outline of the Start-Up Process . . . 63
4.2.2 Running the Controller 67
4.2.3 Arm Calibration Routine 74
4.3 PD Algorithm 75

5. **EXPERIMENTAL RESULTS** 79
5.1 Tuning the Controller 79
5.1.1 PD and PID Controller Tuning 79
5.1.2 Observed "Growling" Phenomenon . . . 93
5.2 Joint Current Measurements 97

6. **CONCLUSION & SUGGESTIONS FOR FUTURE RESEARCH** . . . 102
6.1 Conclusion 102
6.2 Suggestions For Future Research 103

REFERENCES 106

APPENDIX A - HARDWARE NOTES 115
  A.1 System Level Specifications 116
    A.1.1 System Memory Map 116
    A.1.2 VMEbus Slot Assignment 117
    A.1.3 PUMA/Unimate Servo Bus 118
  A.2 Specifications for Project Boards 119
    A.2.1 AMline Decoder Board - VMEbus . . . 119
    A.2.2 Encoder Board - VMEbus 120
    A.2.3 PUMA Interface Board (PIB) - Unimate servo bus 122
    A.2.4 Current Amplifier/Sensor Interface . 122
  A.3 Specifications for U of A / Commercial VMEbus Boards 123
    A.3.1 TUTOR 68000 CPU/RIOT Boards (U of A Developed System) 123
    A.3.2 DATEL DVME601B A to D Board 123
    A.3.3 DATEL DVME628V D to A Board 124
  A.4 Wiring Specifications 125
    A.4.1 VME/Unimation Controller Cable Wiring Lists 125
      A.4.1.1 Encoder Wiring VMEbus Chassis 125
      A.4.1.2 Encoder Wiring Unimate Chassis 126
      A.4.1.3 A to D Wiring VMEbus and Unimate Chassis 127
      A.4.1.4 D to A Wiring VMEbus and Unimate Chassis 128

A.4.1.5 Current Measurement Wiring PIB 129
A.4.1.6 Current Measurement Sensor Board Wiring 130
A.4.2 PUMA Slot #50 Wiring List 131
A.5 Miscellaneous Measurements 133
A.5.1 PUMA Pot Voltage to Index Pulse Resolution 133
A.5.2 PUMA System Status Signals 133
A.5.3 D.C. Resistance of PUMA 560 Joint Motors 134
A.6 System Schematics 135

APPENDIX B - SOFTWARE REFERENCE AND LISTINGS 154
B.1 Software Reference 155
B.1.1 Power Up Sequence 155
B.1.2 Software Start-Up Sequence 155
B.1.3 Calibration and Testing of Boards . . 166
B.2 Program Listings 169
B.2.1 Control.c Listing 170
B.2.2 Skeleton.s Listing 231
B.2.3 User.c Listing 246
B.2.4 Control.spec Listing 247
B.2.5 Sunlink.c Listing 248
B.2.6 Program601b.s Listing 255

LIST OF TABLES

Table 2.1 Comparison of Robot Control Systems 17

Table 3.1 Quadrature State Table 50

Table A.1 VMEbus Board Assignment 117

Table A.2 PUMA Servo Bus Assignment 118

Table A.3 Encoder Wiring VMEbus Chassis 125

Table A.4 Encoder Wiring Unimate Chassis 126

Table A.5 A to D Wiring VMEbus/Unimate Chassis . . . 127

Table A.6 D to A Wiring VMEbus/Unimate Chassis . . . 128

Table A.7 Current Measurement Wiring PIB (DB25) . . . 129

Table A.8 Current Measurement Sensor Board Wiring . . 130

Table A.9 PUMA Slot #50 Wiring List 132

Table A.10 Pot/Index Pulse Resolution 133

Table A.11 Joint Motor Resistance Values 134

LIST OF FIGURES

Figure 1.1 PUMA-Type Manipulator 3
Figure 1.2 Unimate PUMA 560 Robot Controller 4
Figure 3.1 Block Diagram of Control System 36
Figure 3.2 VMEbus Chassis Boards 37
Figure 3.3 Modified Unimate Chassis (Top View) . . . 40
Figure 3.4 Block Diagram of the VEB 46
Figure 3.5 PIB in Unimate Chassis 54
Figure 4.1 Connection Of Multiple Processors 62
Figure 4.2 Controller Main & Communications Menus . . 66
Figure 4.3 Manual Mode Menu 68
Figure 4.4 Test Sensors Menu 72
Figure 4.5 Menu Structure 73
Figure 5.1 Controller Algorithm Performance 80
Figure 5.2a Ziegler Nichols Tuning Joint One 83
Figure 5.2b PID Controller Based on Nichols Tuning Joint One 84
Figure 5.3 PID Controller Improved Tuning Joint One . 85
Figure 5.4 Step Response PD Controller Joint One . . 86
Figure 5.5 Step Response PD Controller Joint Two . . 87
Figure 5.6 Step Response PD Controller Joint Three . 88
Figure 5.7 Step Response PD Controller Joint Four . . 89
Figure 5.8 Step Response PD Controller Joint Five . . 90
Figure 5.9 Step Response PD Controller Joint Six . . 91
Figure 5.10 Ramp Trajectory Joint Two 92
Figure 5.11 Joint Three High Frequency Resonance . . 94
Figure 5.12 Joint Three High Frequency Resonance

Spectrum 95
Figure 5.13 Joint Two Ramp Trajectory, Current Measurement 99
Figure 5.14 Joint Two Control and Current Signals . . 100
Figure 5.15 Joint Two Ramp Expanded View 101
Figure A.1 AMline Decoder Timing Diagram 119
Figure A.2 VEB Timing Diagram 121
Figure A.3 AMline Decoder Board 136
Figure A.4 Encoder Board: Block Diagram 137
Figure A.5 Encoder Board: Quadrature Channel 138
Figure A.6 Encoder Board: Channel Counter & Latch . . 139
Figure A.7 Encoder Board: Main Control Logic 140
Figure A.8 Encoder Board: Clock Divider 141
Figure A.9 Encoder Board: External Connector 142
Figure A.10 Encoder Board: EPLD 1800 Logic 143
Figure A.11 Encoder Board: EPLD 900 Logic 144
Figure A.12 Encoder Board Layout 145
Figure A.13 PIB Block Diagram 146
Figure A.14 PIB: Digital Encoder Buffering 147
Figure A.15 PIB: Motor Servo Command Buffering . . . 148
Figure A.16 PIB: Joint Potentiometer Buffering . . . 149
Figure A.17 PIB Layout 150
Figure A.18 Current Measurement Circuitry 151
Figure A.19 Current Sensing Board Layout 152
Figure B.1 Controller Software Start-Up 157

1. INTRODUCTION

In 1988 the Robotics Control Group of the Electrical Engineering Department purchased a Unimate PUMA 560 robot controller for research and educational purposes. The PUMA 560 robot arm is a six degree-of-freedom robot manipulator driven by D.C. servo motors. Robot motion is detected using potentiometer and optical encoder position sensing. The central rack-mounted controller chassis contains a supervising LSI-11 control computer, separate joint servo computers to control each joint and D.C. current amplifiers to drive the robot arm joint motors. The controller which came with the robot is designed to perform accurate repetitions of simple, taught joint space or cartesian motions. It uses a specially developed operating system and programming language known as VAL II to execute these simple operations. See Figure 1.1 which shows the PUMA 560 manipulator and Figure 1.2 which shows the Unimate PUMA 560 controller.

The Unimate VAL II controller for the PUMA 560 is a simple, inflexible commercial controller. It cannot be reprogrammed since its system code is contained in a central DEC LSI-11 computer EPROM and individual joint servo control algorithms are found in six EPROM programmed Rockwell 6503 microprocessor boards [Fu,et al.,1987]. The VAL II programming environment supports a non-alterable set of robot control and positioning functions. It is difficult to program new real-time control algorithms given the limited trajectory altering functions available. Sensor data transfer must be done through a slow 19200 baud serial link [Unimation,1986] thus complicating the process of adding sensors. Further, the system has a limited trajectory storage space (less than 64 Kbytes) and must perform floating point calculations without a coprocessor. Due to these memory and computing limitations, complex control, sensor, and trajectory planning algorithms cannot be adequately supported by this system. Custom hardware and/or software must therefore be developed to extend its usefulness for more advanced robotics research.

In a flexible manufacturing and robotic research environment a robot controller must be easily modifiable, adaptable, and programmable [Goldenberg and Chan,1988]. It should be able to adapt to changes in control algorithms, external sensor types or operating environments [Ish-Shalom and Kazanzides,1988](e.g. a change from single robot operation to co-operating robots). The controller itself should be easy to modify so that processors, new sensors and improved communication hardware can be added. In addition, in order to avoid considerable redevelopment effort for even minor changes, the controller code should be easy to understand and reprogram. Preferably it should be written primarily in a high-level language. An advanced software

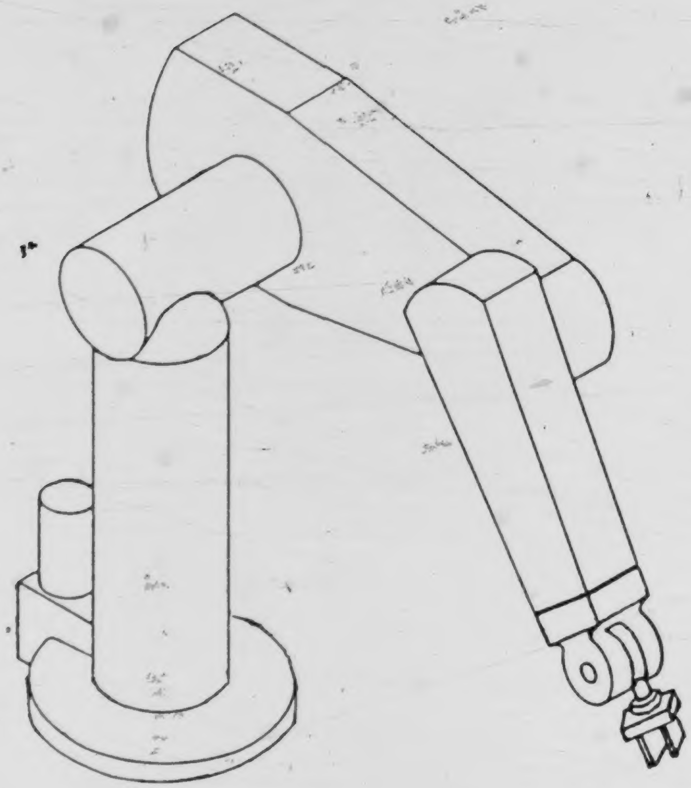


Figure 1.1 PUMA-Type Manipulator

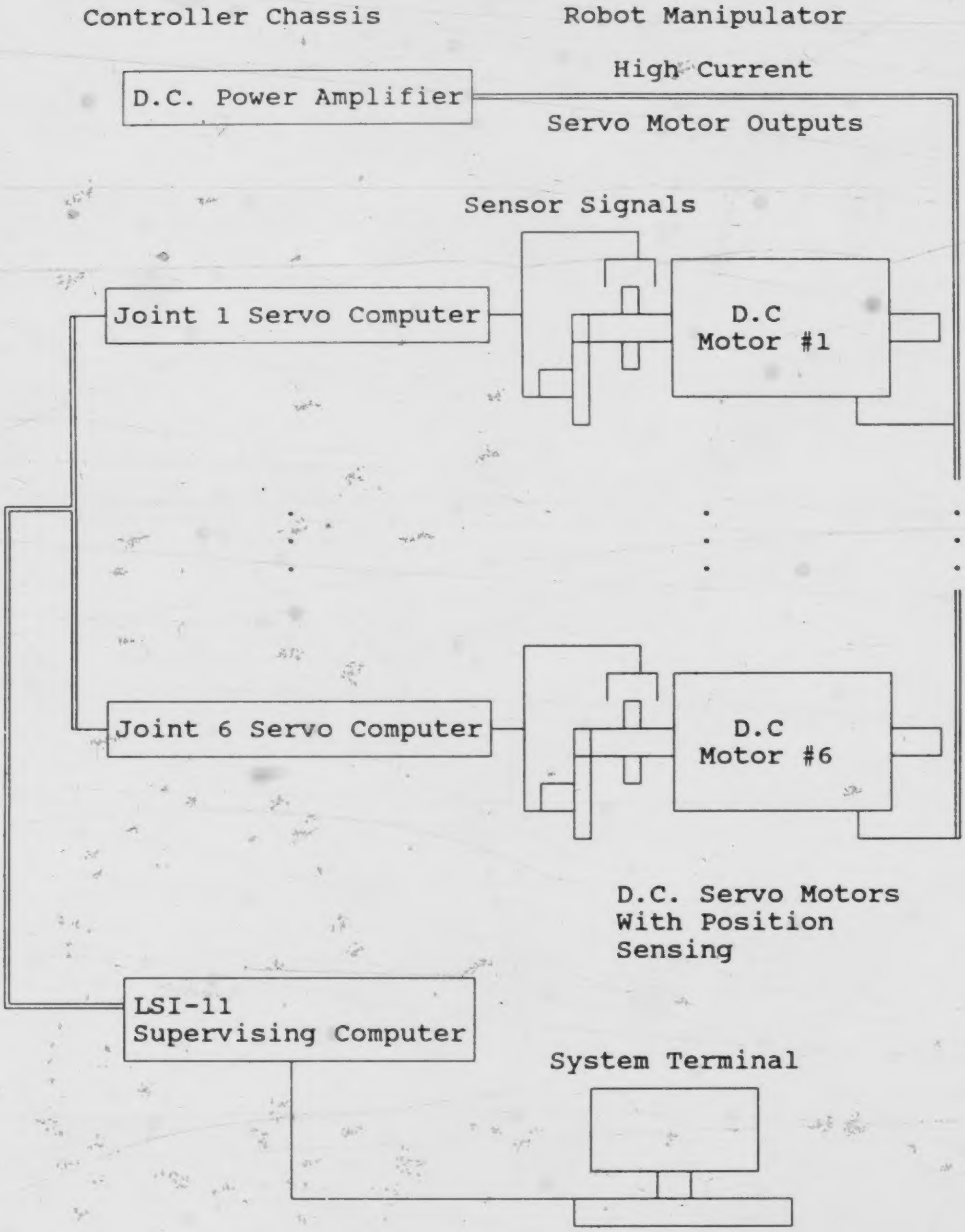

Controller Chassis
Robot Manipulator
High Current
D.C. Power Amplifier
Servo Motor Outputs
Sensor Signals
Joint 1 Servo Computer
D.C
Motor #1
Joint 6 Servo Computer
D.C
Motor #6
D.C. Servo Motors
With Position
Sensing
LSI-11
Supervising Computer
System Terminal

Figure 1.2 Unimate PUMA 560 Robot Controller

development environment is a definite asset [Schmitz,et al.,1989].

To overcome the computational and reprogramming problems of the VAL II controller, the project goal is to create a servo level controller which will be adaptable and flexible. To achieve this goal, the controller will be based on the VMEbus standard so the number of processors can be increased to increase computational power as required. System processors will be reprogrammable to accommodate changes in system requirements. At the servo controller level, this controller will measure not only the position variables directly, but also motor current. With direct access to these variables new control algorithms can be investigated more thoroughly.

The thesis is divided into several chapters. It begins in chapter 2 with a literature review that gives an overview of existing controller architectures and development environments. Also, in this chapter, a somewhat more detailed and specific review of controllers already developed for the PUMA robot is given. In the next two chapters (3 and 4), the hardware and software details of the project are presented. Following this, in chapter 5, the experimental results are presented. In chapter 6 conclusions are made and suggestions for future research are given. Technical data and software listings are included as appendices to the thesis. The technical data include such things as measured robot parameters, wiring tables, and other similar information.

2. LITERATURE REVIEW

2.1 Approaches and Concepts Used in Robot Controller Design

Changes and developments in robotics closely parallel developments in computer and electronics technology. As the latter became more powerful and flexible, so did the corresponding robotic controllers and development systems which could then support increasingly sophisticated features at reasonable cost. In the 1950's and 1960's, with the first British patent in 1954, and the development of the Unimate robot in 1960, researchers began controlling robots with mainframe and minicomputers [Groover,et al.,1986]. In the 1970's, with the development of the single chip computer, microcomputers and workstations were used to control robots. This last development was characterized by a reduction in hardware costs and an increase in computational performance.

Currently, new applications are becoming increasingly specialized and computational requirements continue to increase. In response to this, architectures are becoming more application specific and are using greater parallelism. Increased computational performance is attained using specially developed high performance computers [Anderson,1989,1990] and architectures [Takanashi,et al.,1989]. These systems achieve higher performance using customized arithmetic, memory, and vector processing hardware. At the University of California the RIPS system (Robot Instruction Processing System) has been recently developed. An application specific processor has been developed on a custom VLSI chip to process trigonometric operations in three to four microseconds. The processor is based on an efficient vector rotation, CORDIC algorithm developed by earlier researchers [Butner,et al.,1988]. RIPS is discussed further in the next section.

In conjunction with the construction of "application specific" controllers, researchers are also seeking to boost performance using parallel computer designs. These designs, incorporating multiple medium performance and in some cases "application specific" architectures, can achieve faster controller operation than those controllers based on single, "application specific" high speed processors. Task division (per processor) according to specific algorithms such as kinematics, dynamics, or force control [Butner,et al.,1988][Paul and Zhang,1986] appears to be the trend in current parallel processing research. Task division at sub-algorithmic levels is less feasible in many cases because of high serial dependency within a task [Leung and Shanblatt,1988]. Greater computational performance achieved through parallelism, however, comes with the difficulties of task partitioning, synchronization and communication [Anderson,1989]; consequently for these systems, software complexity and program development times both increase. Parallel processing controllers are discussed in the next section.

Parallel computer designs can be divided into two general categories: homogeneous and heterogeneous systems. Homogeneous systems are constructed primarily around processors of a common microprocessor family. In these systems the software development environment is simplified because no special purpose compilers, cross-compilers, and interface code are required during development. New configurations of processors can be more easily supported since existing software can be ported to newly added processors. Homogeneous systems constructed from commercially available processors are easier to expand and reconfigure.

Heterogeneous systems are constructed using different families of processors. In traditional robot controllers, a central supervising machine coordinated the activity of one or more high performance numeric processors [Schmitz,et al.,1989]. NYMPH is typical of these systems [Chen,et al.,1986]. NYMPH is a Multibus based system in which a central supervising SUN workstation (680x0 family) coordinates the activity of seven NSC32016 processors. This kind of design guarantees efficient and dependable execution of real-time tasks, while at the same time providing a user interface and file storage system. Although recent projects have shown great promise in reducing the development burden [Stewart,et al.,1990] [Butner,et al.,1988], development time is increased for this type of system since special hardware interfaces and customized software [Mangaser,et al.,1989] are needed to connect dissimilar hardware components.

In multiple processor systems the coordination of multiple processes competing for shared resources is handled with various inter-process/processor communication and synchronization strategies. A good reference for these methods is found in [Joerg,1990]. Locally, within individual processor systems, tasks can use local shared memory or local semaphores to exchange data. Local semaphores protect critical code sections and provide synchronization between tasks [Stewart,et al.,1990]. Global shared memory, remote semaphores, and message passing are often available to individual processors. Message passing is frequently used at the global level to provide access to any system resource (e.g. another processor or file server) by any other system element. In some systems, for example, message passing supports the system debugger by passing the state of any system processor to the debugger handler [Narasimhan,et al.,1988,1989]. Finally, at the hardware level, it is very common to synchronize data transfer using hardware interrupts and various types of interrupt exception handlers.

Connected with the integration of many dissimilar processor and input/output components is the principle of hardware transparency. Using device drivers, specific device characteristics are transparently hidden within standardized system level interfaces. If a processor, sensor, or system configuration is changed only specific device drivers need be changed, making system code more portable. Many research controller environments, programmed under UNIX, use a device driver format standardized by the system. Standardized drivers for kernels of heterogeneous processors make different hardware look the same even when it is based on a different architecture or processor family. This standardization frees the programmer of the need to understand specific device or processor details, allowing quicker and more direct implementation of new experimental ideas.

2.2 An Overview of Robot Control Systems

Many different research robot control systems have been developed. To summarize some of the approaches used in leading robot control systems and to review controllers developed specifically for the PUMA 560, this section is divided into three main subsections. The first two subsections establish a conceptual starting point by giving a general overview of the many approaches implemented on present day robot control systems. The third subsection reviews work done specifically on PUMA type robot controllers. Many robot control system features reviewed in these sections have been incorporated into the proposed controller. These features should also be considered as a starting point for future robot projects.

2.2.1 Tabular Comparison

To aid in the comparison of the many different controllers developed, a representative sample of current robot controllers has been assembled in Table 2.1. References for the robot control systems presented in the table are shown below.

Table Number	Reference
1	CHIMERA, Carnegie-Mellon University, [Stewart,et al.,1990], [Schmitz,et al.,1989]
2	SAGE, New York University, [Salkind,1989]
3	RCTS, Queen's University, [Mack and Bayoumi,1989]
4	RIPS, University of California, Santa Barbara, [Butner,et al.,1988], [Mangaser,et al.,1989]
5	Jet Propulsion Laboratory, California Inst. of Technology, [Hayati and Venkataraman,1989]

6	NYMPH, Stanford University, [Chen,et al.,1986]
7	RAL, Rensselaer Polytechnic Inst., [Leahy and Saridis,1986]
8	Portsmouth Polytechnic, [Naghdy,et al.,1988]
9	University of Pennsylvania, [Paul and Zhang,1986]
10	SPARTA, IBM T.J. Watson Research Center, and Brown University, [Ish-Shalom and Kazanzides,1988]
11	INRIA, Domaine de Voluceau, [Dupourque,et al.,1986]
12	RTC, McGill University, [Kossman and Malowany,1987]
13	SIERA, Brown University, [Kazanzides,et al.,1987]
14	TIMIX, University of Pennsylvania, [Lee,et al.,1988]
15	CONDOR, MIT Artif. Intel. Laboratory, [Narasimhan,et al.,1988], [Narasimhan,et al.,1989]
16	University of Toronto, [Goldenberg and Chan,1988]
17	University of California, Davis, [Bihn and Hsia,1988]

18	NEC Corporation, [Takanashi,et al.,1989]
19	AT&T Bell Laboratories, [Anderson,1989,1990]
20	Yale University, [Buhler,et al.,1989], [Whitcomb and Koditschek,1990]
21	Washington University & Jet Propulsion Laboratory, California Inst. of Technology, [Tarn,et al.,1990]

The systems are classified according to whether they have been implemented with homogeneous (HM) or heterogeneous (HT) architectures. The base processor family and bus standard used are also shown. Following this, any special purpose hardware used is indicated and the general method of inter-processor communication between processors in the multiple processor systems is stated.

The vast majority of robot controllers are constructed around the Motorola and Intel line of microprocessors. This is almost certainly because of their price and the availability of software support. Heterogeneous processor systems generally rely on commercially available DSP's (digital signal processors) to boost system performance. The RIPS (CORDIC processor), SPARTA (HERMES), and JIFFE systems are notable exceptions, employing "application specific" hardware customized for robotics applications. Also of interest is the appearance of an INMOS Transputer architecture from Yale University (INMOS T800) which shows promise as an easier-to-use parallel processing architecture.

Inter-processor communication is categorized as either shared memory or message passing type. Shared memory access refers to a simpler hardware level type of data transfer. In the simplest case it means reading or writing to shared memory addresses (asynchronous access). Data protection constructs such as semaphores and asynchronous data buffers [Clark,1989] are also used when synchronization of data access is required. Message passing depends on a higher level protocol and is generally slower than shared memory access because of increased overhead. Messages are passed using a mailbox method which sends commands and pointers to data [Stewart,et al.,1990] between communicating processors. It is generally not fast enough for direct control applications.

	Sys. Type	Base Processor	Bus Standard	Special Hardware	Inter-processor Commun.
1	HT	680x0	VMEbus Multibus	Mercury 3200 FPP TMS 320xx DSP	Shared Mem. Mes.Passing
2	HM	680x0	VMEbus Multibus		N/A
3	HM	80386/286	IBM PC Bitbus		N/A
4	HT	680x0	VMEbus Priv.Bus	Robotic Proc., CORDIC Type TMS320C25 DSP	Shared Mem. Mes.Passing
5	HM	680x0	VMEbus		Shared Mem.
6	HT	680x0	Multibus	NSC32016 DSP	Shared Mem.
7	HM	VAX11/750	VAX-Unibus		N/A
8	HM	680x0	VMEbus		N/A
9	HM	8086/286 VAX11/785	Multibus		Shared Mem.
10	HT	8088	PC bus	HERMES DSP	Shared Mem.
11	HM	68000	VMEbus Smbus G64bus		Shared Mem.
12	HM	8085/86 80286	Multibus		Shared Mem. Mes.Passing
13	HT	680x0	Multibus	Armstrong Multiprocessor Sys.	Shared Mem. Mes.Passing
14	HT	MicroVAXII	Ethernet	Custom Multiprocessors	Mes.Passing
15	HM	680x0	VMEbus		Shared Mem. Mes.Passing
16	HM	NS32016		Custom TUNIS Multiprocessor,	Shared Mem.
17	HM	80286	Multibus		N/A
18	HT	uPD77230 8086	PC bus	uPD77230 DSP based engine	Shared Mem.

Sys. No.	Sys. Type	Base Processor	Bus Standard	Special Hardware	Inter-processor Commun.
19	HT	680x0	VMEbus	JIFFE Robotics Processor	Shared Mem.
20	HM	InmosT800	Serial-Link		Mes.Passing
21	HM	Micro-VAXII			N/A

Table 2.1 Comparison of Robot Control Systems

2.2.2 Comparison of System Architectures

This subsection describes a few of the systems, noted in Table 2.1, in more detail. Comparisons are made and features outlined to provide an overview of current approaches to robot control system architecture. General standard features of controllers and differences in system hardware components are discussed.

2.2.2.1 The CHIMERA Standard

Of all the robot controller philosophies mentioned in Table 2.1, CHIMERA II (Carnegie-Mellon University) is one of the most complete platforms for robot system research. It can be considered a robot controller standard. In published descriptions of this robot operating system one can quickly see many of the trends in current robot control system research development. CHIMERA II began as a homogeneous 680x0 system controller but it is now more than a "controller". It has evolved into a general framework for the integration and concurrent operation of multiple general purpose processors, multiple computer backplanes, special processors, and general sensory and control devices. Device and processor differences are hidden by standard driver interfaces and kernels and a complex communication layer. This allows commercial and special purpose hardware to be integrated more easily.

CHIMERA II is based on the NASREM model proposed by the National Institute of Standards and Technology, U.S.A.. Elements of this model are present in most research controllers. Robot systems and hence programs are by nature hierarchical [Clark,1989] and hence NASREM provides a hierarchical model for the efficient integration of multiple sensors and robots into a single application. The NASREM model hierarchy is built upon a series of levels. Each level has sensory processing modules (responsible for obtaining and integrating information about the system), world modelling modules (organizes and controls access to the global database), and a task decomposition modules (executes tasks based on user input and information). Six levels of these modules are suggested by the standard, but in a practical system this might be simplified to three levels as suggested by Mack and Bayoumi. (See the next subsection dealing with "Other Systems" for a more detailed description)

To support task execution, CHIMERA II also contains fundamental real-time software constructions similar to those found in many other research controllers. Like many other systems it supports standard inter-processor/processor communication and synchronization primitives and device drivers. In addition, multiple options for task priority scheduling provide added real-time task scheduling flexibility.

Priority can be scheduled in three ways: (1) minimum-laxity-first (mlf), (2) highest-priority-first (hpf), and (3) round-robin (rr). Mlf scheduling assigns a deadline time and tries to run all tasks before their deadline. The tasks with the minimum remaining execution time (minimum laxity) are given highest priority. If the system is too busy and cannot run all tasks, a failure handler runs selected tasks and/or degrades performance of other tasks to try to meet priority task deadlines. Tasks may also be scheduled from highest to lowest priority when multiple tasks have equal laxity or do not have deadlines. Equal priority tasks can be run in a circular queue arrangement called round-robin priority. Task priority may also be inverted under certain conditions.

2.2.2.2 Other Systems

The RCTS controller software uses a simplified, yet powerful subset of the NASREM model to organize robot control modules (refer back to the CHIMERA II section for NASREM definition). Controller processing levels, based on this subset, can be directly and easily implemented in any given research robot control system. In the RCTS controller, the lowest controller processing level obtains sensor information and calculates and sends the drive signals to the servos. This level is given highest priority, in a descending fashion, over higher levels so that its real-time tasks may "pull down" robot commands from higher levels. This ensures efficient and fast low level execution. The middle level puts sensor data into a qualitative form (e.g.payload weight, center of gravity etc.) and calculates the set points to be sent to the controller. It is not as deadline dependent as the lowest level. This trajectory planning level might be driven by inverse kinematics, dynamics, or other sensor based calculations which are coordinated by a highest or task planning level. At the highest level, decisions are based upon abstract data constructs related to tasks such as picking up a part moving in space.

RIPS is a heterogeneous system designed to flexibly support numerous multiple manipulator, processor and sensors configurations. To provide the necessary computational power, "algorithm" or job level parallelism is incorporated into a robotic processor which uses two custom CORDIC processors (trigonometric function calculation) to calculate inverse kinematics and dynamics. These computationally intensive algorithms are run in parallel, thus saving execution time. To further speed execution, hardware is also used to implement real-time synchronization and communication for asynchronous data transfers between servo and algorithm level processors.

RIPS is also significant because it simplifies the traditional problem of integrating an easy to use software environment on top of a concurrent heterogeneous architecture. Using fast hardware, it retains great computational speed while maintaining its ability to easily and naturally express and test control problems. Other systems such as CONDOR have not been able to or have not attempted to maintain such a favourable balance.

The CONDOR controller system is an elegant architecture which is computationally very efficient, flexible, and realistic for research institutions with limited funding and technical support. It uses a single VMEBus, multiple 68020's, a SUN 3 host and no potentially expensive custom hardware. The proponents of this system believe that robot control algorithms can be most naturally divided into large tasks suitable for individual processors. Tasks are thus allocated by the user rather than by a task scheduler. Another advantage of this homogeneous architecture is that additional computational power can be gained by adding more processors. Because it is based on similar processors, this system can support inter-processor communication using a simple and very fast interrupt driven message passing strategy. As a result of its architectural simplifications, CONDOR avoids the many layers present in more transparent, flexible and hence less efficient controller architectures. This simplicity, although an advantage for smaller systems, is also a disadvantage. As the system grows, bus contention problems between processors limit maximum achievable performance.

The robot control systems constructed around the JIFFE processor (based on the AMD29331 program sequencer) and the uPD77230 are typical of the high performance, special purpose computing engines found in robotics research. These processors are tailored to perform vector/matrix and other similar calculations, typically found in robot algorithms, at extremely high speeds. These systems use high performance commercial processors, thus avoiding the development time needed to create more application-specific custom processors such as CORDIC. Overly specialized processors gain performance on a very specific task but may lose their speed advantage in overall process computation as data must be preprocessed to get it into a suitable form [Anderson,1989]. A change in the target algorithm may further decrease efficiencies as the problem solution and computational architecture become less compatible [Butner,et al.,1988]. Another advantage of these high performance, DSP (digital signal processor) based systems is that fewer processors are needed to achieve the equivalent computing capacities of more sophisticated parallel processing architectures. Parallelism in multiple processor systems greatly complicates system development.

High performance DSP based systems have similar disadvantages to application-specific processors and cannot escape (although they extend) the computational limits imposed by any processor technology. DSP systems, being quite specialized, often require longer software development times because they lack the powerful development environments coming with more standard processor family systems. DSP's may also have to emulate more complicated arithmetic operations (e.g. trigonometric and floating point calculations), slowing their operation for specific algorithms [Hanselmann,1987]. Standard coprocessors perform these operations more efficiently. Finally, the reduction in the number of processors required to perform a task is only a short term gain made possible by the specific DSP used. As the demand for computing power continues to rise with more complicated robotic applications and algorithms, its seems that these powerful processors, designed to avoid parallelism, will inevitably become smaller building blocks in newer, more complicated parallel designs.

An innovative hardware solution investigated by Yale University easily harnesses the power of parallel processing. A general set of custom processor boards based on the Inmos T800 were developed and configured as required on a simple common I/O bus. The inherent concurrent and non bus-based nature of the Transputer hardware architecture and software development environment makes the investigation of parallel architectures much easier. Time consuming development of custom kernels, distributed communication and task allocation schemes of conventional common bus-based schemes is apparently avoided as much of this is done by the hardware. The bus contention problem of conventional bus-based schemes is also avoided using the unique, multiple, high bandwidth serial data connections found on each Transputer. Serial message passing is currently being investigated to evaluate its potential value for real-time robot control system design.

Although current systems have widely diverse features there are several common trends in robot controller system design. All the systems which have been reviewed here are based on hierarchical software and hardware designs. These designs are necessitated by the hierarchical nature of the robot system plant. Most systems share or have a subset of common low level primitives mentioned for priority scheduling, communication and synchronization. More specialized hardware is being developed to compute algorithm level concurrent tasks more efficiently. Similarly, more parallelism is being used as the computational demands rise for newer, increasingly complicated robot applications.

2.2.3 Robot Controllers for the PUMA 560

In the previous portions of this subsection a general review of robot control systems was made. In this portion of the subsection controllers specific to the PUMA 560 are described. As an introduction to this description, the Unimate PUMA 560 controller is discussed first in more detail to illustrate the possible access points for controller modification.

The PUMA controller can be modified in several ways so that an external computer system may control its operation. Depending on the method chosen the researcher can control the PUMA at the proprietary operating system, or VAL II, level; at the Unimate controller bus level, or at the servo level. Specific attention is given to those controllers which control the PUMA at the servo level as is the case for this project.

To aid in the review of existing PUMA research controllers a brief hardware description of the PUMA controller will be useful. As outlined in the introduction, the PUMA controller is based on a DEC LSI-11 Q-Bus computer which is connected to six servo control computers on a servo bus. Each servo computer controls a separate joint using a fixed PID controller. Communication between the DEC LSI-11 Q-Bus and the servo computer servo bus is done through cabling from A to B interface boards connected on each of the respective buses. External communication to the system can be done through a RS-232 terminal port, a 9600 baud disk drive port, a SUPERVISOR port (9600 baud), an ALTER port (19200 baud), or directly through the Q-bus or servo control buses [Unimate,1985]. Position, servo control and motor current signals can be intercepted and altered at specific locations within the controller.

By modifying or replacing the existing PUMA controller hardware and writing new software, the PUMA 560 robot arm can most effectively be used for new research and teaching purposes. There are four practical methods by which custom hardware can take direct control of the PUMA controller [Goldenberg and Chan,1988]. In the first method the DEC LSI-11 Q-Bus is controlled externally through a DRV-11J Parallel Interface board available from DEC Corporation. In this approach the existing LSI-11 and its EPROM memory must be disabled to avoid conflicts with VAL. The second method separates the PUMA LSI-11 VAL controller from the six servo processors by disconnecting the A and B interface boards (DRV-11-J parallel boards) and inserts an alternate processor. This processor then directly takes control of the six servo boards. This method was used by researchers at the Rensselaer Polytechnic Institute (RAL) and the University of Toronto. Another method adapts the ALTER serial port to communicate with an external computer and alters or replaces the existing DEC LSI-11 computer so that it can receive external command signals. These signals are used by the modified LSI-11 to directly control the PUMA joint servos

[Goldenberg and Chan,1988].

The fourth approach controls the robot arm servo amplifiers directly by bypassing the DEC LSI-11 and Rockwell 6503 boards completely. This method has been used in several systems such as SIERA, [Paul and Zhang,1986], [Bihn and Hsia,1988] and [Tarn,et al.,1990]. The fourth method requires more custom hardware than the other methods, but allows direct, unrestricted control of the robot arm servo motors. It met the requirements of this project, i.e. having direct access to servo level signals, and hence was the approach chosen.

Each of the systems, accessing the controller at servo level, are slightly different. The earliest servo level controller found in the literature used Multibus to connect multiple 8086's in what was called a force/motion server [Paul and Zhang,1986]. The SIERA project, using a special purpose ARMSTRONG processor and Multibus to connect 68000 processors and the robot interface, was announced in the next year. This controller has a more advanced operating system. The third system, a universal six joint robot controller, is based on a 80286 processor and Multibus interface to the PUMA robot [Bihn and Hsia,1988]. Very recently a fourth system based on a MicroVax II was connected to the Unimate controller with a "Motion TEK" robot controller [Tarn,et al.,1990]. Of the few details yet released about this system, its servo motor current monitoring feature is the most interesting.

2.3 Comparison of Software Environments

To develop a successful robot control system the computer operating system and associated development, debugging and analysis software for the controller must be chosen carefully. This section, based on the reviewed literature, serves as a guideline for future software development and purchases. What is good in one system may not be practical or possible in another system. Too many or too complicated features in controller software will degrade system performance unless more expensive configurations of processors are used to support the additional features. The RCTS controller software, for example, was organized by the convenient NASREM standard, but this contributed to reduced control algorithm execution times. Robot software design should be based on a balance between performance, system complexity, and cost.

CHIMERA II can be used as a starting point for discussion. This robot control system, like many other systems, was developed around an existing non-real-time operating system (NRTOS), UNIX. This type of general NRTOS provides the development environment and file server support to design, test, and run a system. Additionally by using standard operating systems and their programming languages, pre-developed robot motion/control packages such as RCCL

[Hayward and Paul, 1986] can be directly used in new systems, cutting development time.

Real-time kernels such as those used in the SAGE and TIMIX projects provide alternatives to or enhanced features for standard operating systems. These kernels coordinate or supply system debugging support, memory management, network communications access, file server support, and exact timing structures required for real-time supervisory applications. Unfortunately, intensive software development is required to duplicate a subset of the more general commercial operating system features. This type of development is not required for typical research applications, as existing NRTOS's can be adapted (by modifying system drivers, libraries and development tool primitives), in a less time-consuming process, to fit into real-time robot controller architectures. Further these real-time kernels which have higher level supervisory capabilities are useful in much larger multiple manipulator, factory-level coordination tasks and are generally beyond the financial scope of university projects.

In considering operating systems for non-real-time development environments, one should also investigate purchasing a commercial real-time operating system (RTOS). A commercial RTOS could save on the development time needed to create such RTOS constructions as priority scheduling, communication, and synchronization primitives. These and other useful primitives are often standard features of such operating systems [Davis,1986] [Small,1988]. A real-time controller system could thus be built upon a commercial RTOS and run in conjunction with a standard NRTOS host as is done with several of the custom, research real-time operating systems.

Commercial RTOS's have been tried by several research groups with mixed success. This approach has been tried by Naghdy [Naghdy,et al.,1988] in a system containing a Mitsubishi robot controlled by a VMEbus based computer. In the RTC project, a similar commercial operating system is used to control a Multibus based system. In both cases the system performance was quite slow. Also with present commercial RTOS's much time is spent developing robot specific features not already present within the operating system. Making code level modifications to suit the application is unfeasible since source code is often not available or very expensive. Present evidence suggests that commercial RTOS's are not entirely suitable for robotic applications.

The choice of development language used can also directly affect a system development and execution time. It is advantageous to transparently hide the lower levels of software and hardware from the user. This frees the programmer from thinking about low level details and hence makes achieving a research goal that much easier. Lower level details can be hidden using operating system drivers which are directly accessible from the programming language chosen. "C" language under UNIX is ideal for this purpose. The structure of the language itself can directly influence transparency. Object-oriented languages such as C++ allow a level of abstraction separating the level of specific manipulator motion from higher level robot tasks [Mangaser,et al.,1989]. Where speed is a critical factor, assembly language should be used to optimize execution times. One should restrict the amount of assembly code used as it is difficult to develop and modify.

Special development and display tools allow for the easiest and fastest development, testing, and debugging of code and analysis of data. During the initial development at the device level, a system monitor is indispensable for testing and debugging of hardware. Such a monitor must have routines to single step through code, to set break points, and to allow direct access to devices and memory. At the higher programming level, compilers and cross-compilers should ideally be supported by symbolic debuggers. Time analysis routines are essential for real-time development since it is rarely obvious what the real-time characteristics of a system or program may be. Simple stop-watch routines such as tick(), start timer, and untick(), stop timer, [Kernighan and Plauger, 1974] are essential for developing real-time systems.

A user friendly display of system status data, programs, and experimental data during and after development can greatly ease system use. Window interfaces are suggested in the literature for several robot control systems such as RIPS, NYMPH and CONDOR. They can flexibility display multiple processes on one screen. The need for this type of display interface is more critical for multiple processor systems where increased numbers of concurrent processes must be monitored. Using windows, data coming from an experiment can be pre-processed and analyzed directly in the context of the current dynamic system configuration. Editing files, testing software, and changing controller settings can be done quickly and conveniently from the same terminal [Chen,et al.,1986].

3. DESCRIPTION OF PROPOSED CONTROLLER HARDWARE

3.1 Design Goals and Methodology

The proposed controller for the Unimate PUMA 560 controller evolved in a series of specification, prototyping and construction phases. To avoid confusion in the following discussion, the proposed project controller will be referred to as the VMEbus controller to distinguish it from the Unimate controller which is the original VAL II based controller provided by Unimation corporation.

At the beginning of the project, several existing systems were researched and their beneficial characteristics were identified. Servo level controllers for the PUMA 560 robot (see literature review) were closely examined to assist in the definition of a servo level interface. Technical details such as sensor and servo signal access points were identified from the descriptions of these systems and from the Unimate technical manuals [Unimation,July,1985]. Using this information an input/output specification for the system was created and used in the design of system interface boards.

Features from more general robot control systems were also incorporated into the design. Modular and expandable systems such as CHIMERA II and RIPS are successful because they are reconfigurable for different manipulators, sensors and additional processors. Efficiency and simplicity are also desirable. CONDOR is a simple yet powerful system based on multiple processors connected to the VMEbus. Based on comparisons to these and similar systems, it was decided that the VMEbus would be an efficient and expandable platform for present and future processor and sensor configurations. Using the VMEbus, new custom hardware could be easily integrated. Also, availability of a powerful development environment based on SUN and HP software packages and a custom University of Alberta TUTOR 68000 processor (described later) as well as a twenty one slot, stand alone VMEbus chassis were important considerations.

Many of the surveyed systems divide non-real-time and real-time functions between physically separate processor groups. In CONDOR and NYMPH, a host SUN development system (non-real-time) is connected to a multiple processor system which executes servo level and other controller real-time tasks. This type of division is also used for this project (refer to Figure 3.1). The HP9000 and SUN 3/160 computers are used for program development and file storage. The SUN 3/160 running under UNIX is unsuitable for real-time applications because a predictable real-time sampling interval cannot be guaranteed. The VMEbus chassis computers and the PUMA 560/Unimate LSI-11 computer run the real-time robot controller software. The sampling interval can be guaranteed in the VMEbus chassis computers since they do not have non-maskable, operating system controlled interrupts.

3.2 System Configuration

Referring to Figure 3.1, one can get an overview of the system. Programs are developed on the SUN 3/160 or HP9000 UNIX systems, and then are cross-compiled on an efficient HP9000 compiler/downloader program. The resulting S-records are transferred through Ethernet to the SUN 3/160 then moved along the RS-232C #1 to the VMEbus chassis for execution by the processors contained in the chassis. Real-time program operation is controlled using RS-232C #2 connected to a separate "main terminal". The Unimate chassis is connected to the VMEbus chassis by the PUMA interface board (PIB) and associated shielded, twisted pair cables. An internal servo motor current measurement board is also connected to the PIB. RS-232C #3 connector is used to communicate with the Unimate controller VAL II operating system during initialization only.

The VMEbus chassis shown in Figure 3.2 contains a main CPU and several input/output boards (for card cage board assignment within the VMEbus chassis see appendix A). The main computer based on the 68000 processor was developed by the University of Alberta Electrical Engineering Department. This board is referred to as the TUTOR 68000 main CPU and its accompanying serial interface board as the TUTOR RIOT Board (RAM I/O Timer board). TUTOR refers to the modified Motorola

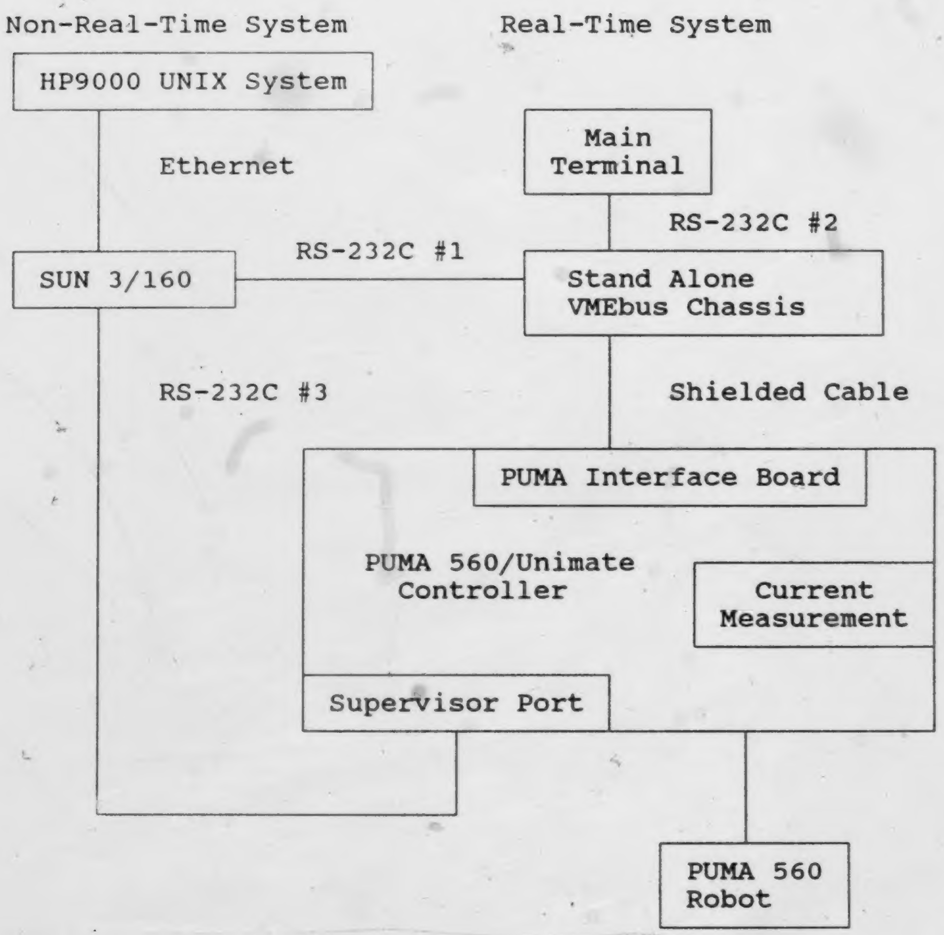

Non-Real-Time System
Real-Time System
HP9000 UNIX System
Ethernet
Main Terminal
RS-232C #2
SUN 3/160
RS-232C #1
Stand Alone VMEbus Chassis
RS-232C #3
Shielded Cable
PUMA Interface Board
PUMA 560/Unimate Controller
Current Measurement
Supervisor Port
PUMA 560 Robot

Figure 3.1 Block Diagram of Control System

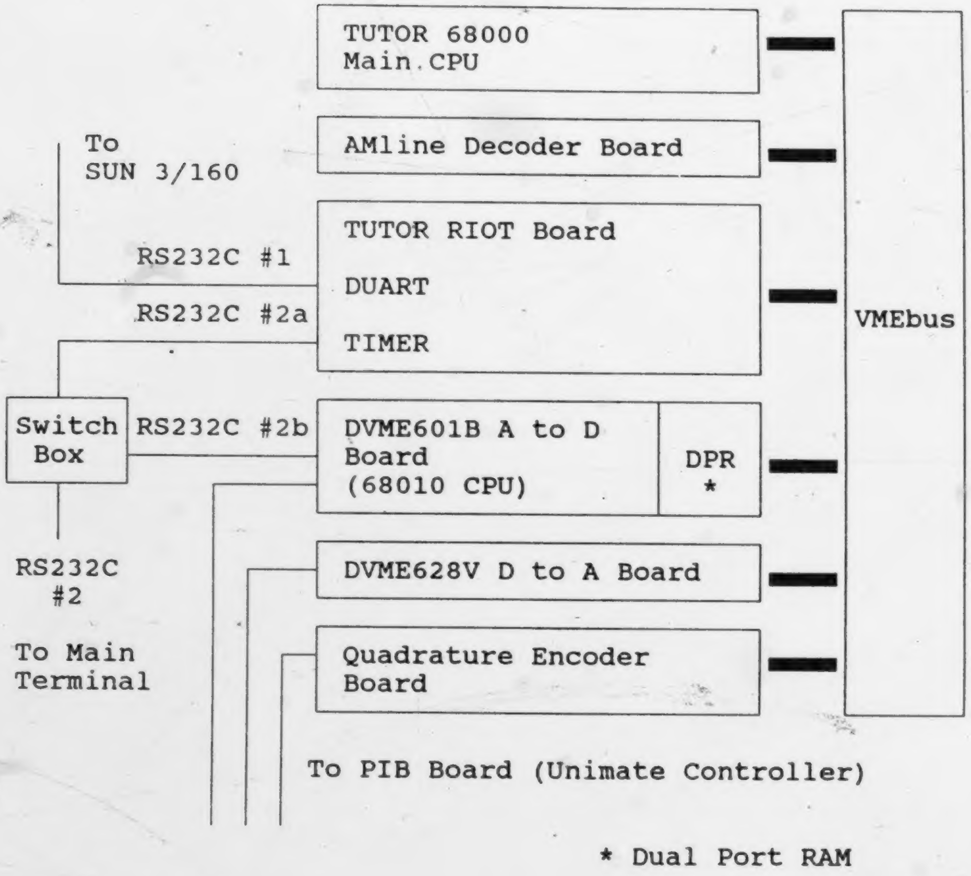

TUTOR 68000
Main CPU
AMline Decoder Board
To
SUN 3/160
TUTOR RIOT Board
RS232C #1
DUART
RS232C #2a
TIMER
VMEbus
Switch
Box
RS232C #2b
DVME601B A to D
Board
(68010 CPU)
DPR
*
RS232C
#2
To Main
Terminal
DVME628V D to A Board
Quadrature Encoder
Board
To PIB Board (Unimate Controller)
* Dual Port RAM

Figure 3.2 VMEbus Chassis Boards

TUTOR monitor program which runs the main CPU. This CPU performs most of the controller functions and monitors the main terminal for user input. The TUTOR RIOT board operates RS-232C lines for external devices and runs a timer which serves as the system clock. The DATEL DVME601B analog to digital converter/coprocessor board contains a 68010 processor which pre-processes signals coming from the PUMA 560 controller joint potentiometers, current measurement circuit, and system control lines. An RS-232C #2b connection is made from this board to the main terminal (switch box selected) to observe the 601B monitor program status during debugging and system operation. Next to this is the DATEL DVME628V digital to analog converter board which sends servo motor signals and an arm power "on" enable signal to the PUMA 560 controller.

Two boards, the AMline decoder board and the quadrature encoder board, custom built for this project, are also located in the VMEbus chassis. The AMline decoder board is used to improve the TUTOR 68000 system to meet VMEbus revision B standards [VMEbus Specification Manual,1982]. This board operates the address modifier lines number three to five which are not controlled directly by the TUTOR 68000 system. The quadrature encoder board is used to process quadrature encoder and index pulses signals coming from all six joints of the PUMA 560 robot. Under main CPU command this board will simultaneously latch absolute encoder position count values for all six joints or will initialize counters individually upon receiving joint index pulses from a selected PUMA 560 robot joint.

The Unimate chassis interface of the VMEbus controller is shown in Figure 3.1 (for card assignment on the Unimate servo bus see Appendix A). The PUMA interface board (PIB) is connected internally to the Unimate chassis servo bus at slot number fifty. The PIB provides signal conditioning and isolation between the Unimate and VMEbus chassis portions of the project controller. Encoder, joint potentiometer, arm power enable, system status monitoring, and servo motor command signals are wired from other areas of the Unimate chassis to this bus slot. In this way the Unimate controller is bypassed. In addition, a cross-chassis cable connection is made to a current sensor interface found directly above the Unimate motor current amplifiers. The current amplifier subsystem and current sensor interface presently monitor the servo current going to robot joint two. This interface has been designed to accommodate current monitoring for joints one and three also, but this is left as a future expansion option. The layout of the Unimate chassis with these modification is shown in Figure 3.3.

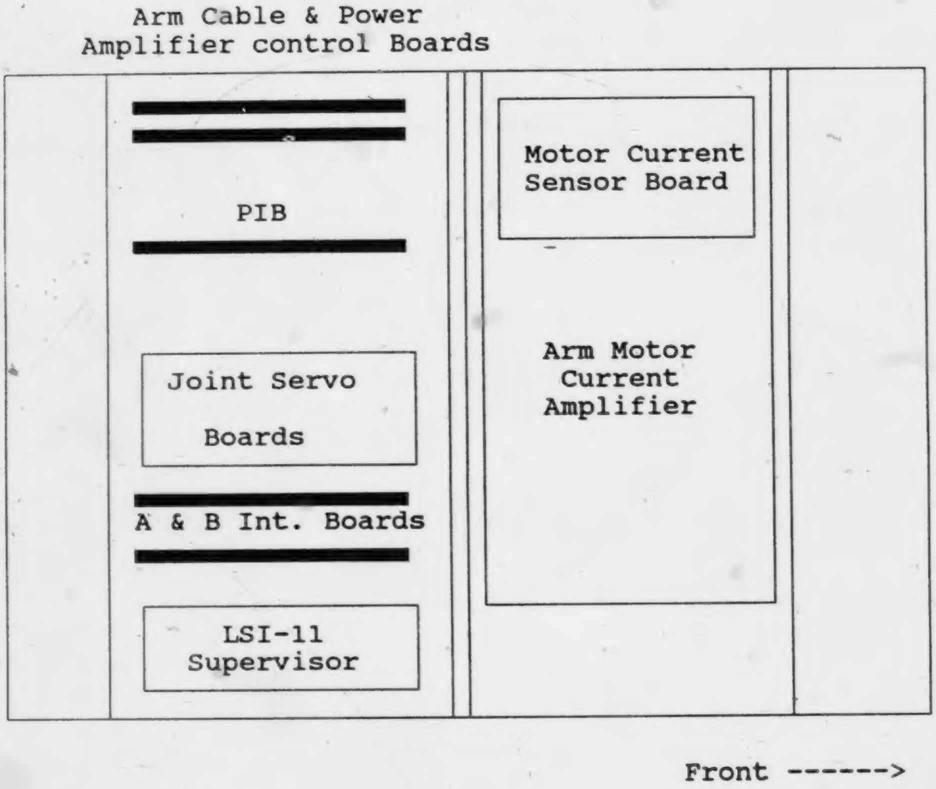

Arm Cable & Power
Amplifier control Boards
PIB
Joint Servo
Boards
A & B Int. Boards
LSI-11
Supervisor
Motor Current
Sensor Board
Arm Motor
Current
Amplifier
Front ------>

Figure 3.3 Modified Unimate Chassis (Top View)

3.3 Description of Individual Boards

In this section, individual boards purchased or constructed for this project are described. Before this is done the principles and guidelines used in the design and construction of these boards are given.

3.3.1 System Design Decisions and Goals

At the beginning of the project several development decisions were made. It was felt that simultaneous development of hardware and software would be undesirable since one would never be certain whether a problem is hardware or software related. Hardware was thus developed and completely tested prior to any significant software development. To avoid ambiguities as to which signals were assigned to a given board, device and system interfaces were pre-defined according to functional units containing groups of similar or closely interacting signals.

Related to these development decisions, design goals were established to ensure consistency in the design process and to make the resulting system easier to test and maintain. The system was to be as flexible as possible incorporating such features as jumperable control line and address selection. To maximize performance, system hardware was to be built to minimize computer access and polling cycles. All construction was to be done using wire wrap connections as this is easy to modify and is also one of the most reliable, fastest and cheapest construction methods for small quantities of circuit boards [Horowitz and Hill,1989]. As much as possible all system parts such as integrated circuits, electronics, connectors and even boards were to be of the "off-the-shelf" variety. Custom parts are often expensive, difficult to obtain on short notice, and frequently go out of date. These problems delay construction and complicate later maintenance.

At the device level several additional standards were also maintained. Interfaces between systems were buffered and shielded whenever possible to isolate the effects of device transients and failures, to reduce system noise, and to reconstruct degraded signals. Signals were made fail safe or fool sensitive where ever possible. For example, power up default signal levels were selected to be as safe as possible, fusing and similar zener diode power supply protection were included, and main critical switches were placed in less accessible places to force the user to think before switching them.

3.3.2 Commercial Boards

Two commercial boards, the DATEL DVME601B analog to digital converter board and the DVME628V digital to analog converter board were purchased. Complete documentation for these boards is found in their individual manuals.

The DVME601B board was modified and a special program downloaded to its local memory. This board uses an on board 68010 processor to pre-process analog to digital conversions triggered by external start scan signals. A timer chip on this board is used to generate a periodic sampling pulse. This signal is fed back by a jumper which is connected to its multi-function peripheral I/O connector. This modification causes an external analog to digital sampling trigger. Periodic analog samples, synchronized in this manner, are filtered using averaging and selected channel samples are monitored for level errors. This pre-processed signal data is then collected by the main processor. The pre-processing burden is not placed on the main processor and as a result overall system throughput is increased.

3.3.3 AMline Decoder Board (ADB)

The address modifier line decoder board controls the address modifier lines AM3..5 on the VMEbus. These lines cannot be controlled directly by the TUTOR 68000 system. Address modifier lines are responsible for switching between VMEbus addressing modes. The standard and short addressing modes are used by the project VMEbus boards. In the first mode all twenty four address lines on the P1 VMEbus connector are used to access a VMEbus board. Short addressing uses only the first sixteen address lines and thus can address a sixty four kilobyte range. Some slave boards (only the DATEL DVME 628V in the project) use this type of addressing because they can have simpler address decoding logic.

The ADB decodes a byte pattern supplied to it and uses it to control AM lines three to five. When a new address modifier mode is required, the code for this mode is latched to the decoder immediately before address accesses are executed. At the end of an addressing cycle in this mode, the address modifier code for the next addressing mode is latched by the ADB. Addressing in the new mode can then begin in the next clock cycle.

The operation of the ADB is divided into a decode and latch sequence. On a given cycle, the proper address for the ADB is placed on the address bus and the address strobe line (AS*) is driven low (refer to APPENDIX A for schematic). The decode section of the ADB circuitry then produces a pulse which makes the first four data lines accessible to the 74173 latch. In a sequence of two writes in which data line three is pulled high then low, data is latched by the 74173 and the VMEbus AM lines three to five are set to the signal levels corresponding to the bit pattern found on data lines zero to two. The state of the address modifier lines is thus controlled by the user in a sequence of two data writes.

3.3.4 VMEbus Encoder Board (VEB)

3.3.4.1 VEB Introduction

This VMEbus encoder board (VEB) was developed since commercial quadrature encoder boards with suitable features were not available. Surveyed boards were too expensive and did not have simultaneous encoder count sampling.

The VMEbus encoder board (VEB) records relative robot joint location by counting the optical encoder pulses sent by each joint of the PUMA 560 robot. These pulses and the direction of robot arm rotation are encoded in quadrature signals coming from each joint. This signal is converted into up or down counter clock pulses which indicate whether the arm has rotated in one direction or the other. It is the function of the VEB filter chips to decode the quadrature signal into these counter clock pulses. Index pulses, also sent by each robot joint, are used to initialize the starting count value of each of the encoder counters.

The block diagram shown in Figure 3.4 shows the structure of the VMEbus encoder board. The VEB has two main functional components: the VMEbus interface and the quadrature decoding circuitry. VMEbus decode and control logic found in the central EPLD 1800 (refer to schematics in Appendix A) performs jumper-selected address decoding and control line functions required for a standard slave type

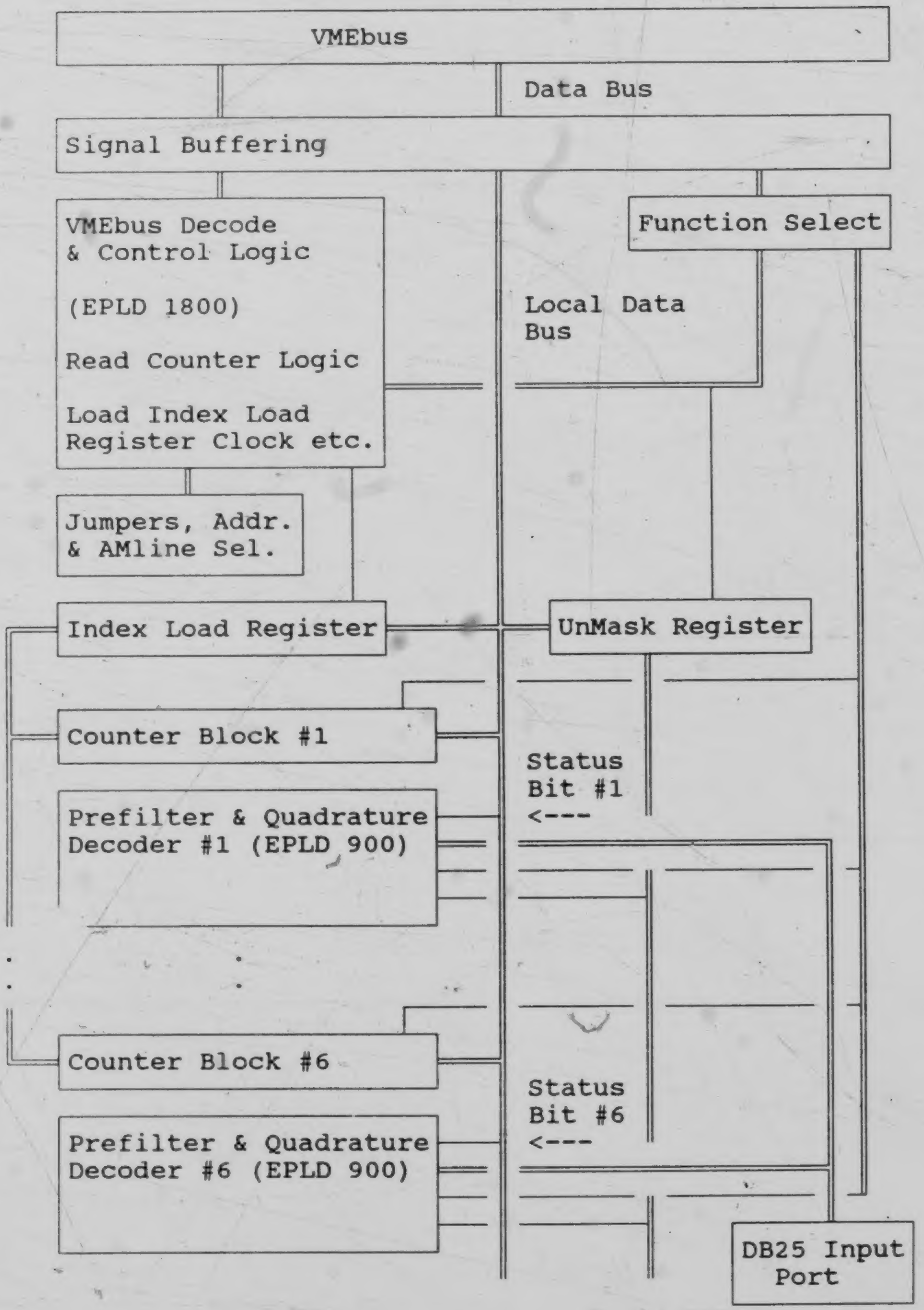

VMEbus
Data Bus
Signal Buffering
VMEbus Decode & Control Logic
(EPLD 1800)
Read Counter Logic
Load Index Load Register Clock etc.
Function Select
Local Data Bus
Jumpers, Addr. & AMline Sel.
Index Load Register
UnMask Register
Counter Block #1
Status Bit #1 <---
Prefilter & Quadrature Decoder #1 (EPLD 900)
Counter Block #6
Status Bit #6 <---
Prefilter & Quadrature Decoder #6 (EPLD 900)
DB25 Input Port

Figure 3.4 Block Diagram of the VEB

VMEbus board. This EPLD also contains the control logic necessary for latching, reading and initializing encoder pulse counters.

The quadrature decoding circuitry is implemented with the remainder of the logic circuits on the VEB. The function select block driven by the four lowest address lines selects which of the counter read, index pulse unmask, counter clear, and status read functions are to be performed upon a given encoder board access. The index load register is used to initialize the count value of a selected counter upon the arrival of an index pulse. The unmask register is used to select which counter is initialized by an arriving index pulse. The status register, whose bits are supplied by the six prefilter chips, records when such a selected pulse arrives. The six counter blocks contain four, four bit 74LS193 synchronous up/down counter chips and their output data latches. The six prefilter and quadrature decoder blocks each contain a single custom EPLD 900 filter chip. Encoder signals are sent to these filter chips through a DB25 connector located on the board edge.

3.3.4.2 VMEbus Interface Design

To connect the VEB to the VMEbus it was necessary to develop a VMEbus interface. It was decided that a simple slave type interface would be the simplest way to implement the design since a minimum of bus arbitration and no interrupt handling (Interrupter) circuitry is needed. This interface includes two groups of VMEbus compatible buffer chips and an EPLD 1800 which provides address decoding, control line, and counter read/write functions.

In a VMEbus system different types of buffers must be used at the bus interface because of electrical differences between different classes of control, data and address lines. Control, data and address lines use standard 74LS245 tristate logic buffers. An exception, the DTACK* line, used to signify a successful address access, is buffered using an open collector 7438.

The EPLD 1800 chip contains the logic necessary to implement the VMEbus interface (refer to the EPLD 1800, VME Interface schematic in Appendix A). Board addressing is implemented by comparing board address select and address modifier select jumper signals to incoming address and address modifier lines. The address select line coming from the EPLD 1800 is used to enable the data bus buffers and the function selection demultiplexer (74LS154). The delayed DTACK* signal is derived from the address select signal. The load select logic controls reads of each individual counter channel and allows encoder counter initialization data to be written to the index load register.

The VMEbus interface control line timing can be summarized as follows (refer also to Appendix A, VEB timing diagrams). Address decoding logic quickly enables the local data bus and board function selector. Next, a delay shift register and an internal D flip flop generate a timed sequence of VMEbus and internal control signals. The DTACK* signal is generated shortly after a valid address has been selected. During encoder counter initialization, the index load register data-latching clock pulse (INDXCLK) is also delayed slightly to give the load select logic a chance to settle while new data is being written to the index load register. Finally the delayed function selector off signal is generated late in an access cycle so that internally selected logic may have time to execute. The delay also guarantees that erroneous board signals are not generated.

3.3.4.3 Quadrature Decoder Circuitry Design

The quadrature decoder design implemented on the VEB can be divided into two separate parts. The first part involved designing prefiltering and quadrature decoding circuitry. The second part of the design involved developing an index pulse driven, counter initialization circuit.

The quadrature decoder part of the project began with the selection of commercial counters to be used to count encoder pulses. Three counters, 74LS569, 74LS191 and 74LS193 are locally available. From the point of view of cost, reliability, and flexibility, 74LS193 was selected.

Quadrature encoding is a method of storing information about relative position and direction of rotation of a shaft. This information is stored in the form of two, out of phase quadrature signals. For discussion, call these signals channel A and channel B. All possible channel A and channel B on and off states corresponding to the clockwise and counter clockwise rotation of the joint encoder can be summarized in a simple sixteen state table. Counter-clockwise rotation of the joint encoder is associated with a clock decrement and clockwise rotation with a clock increment. ' and " markings correspond to present and previously sampled states of the quadrature channels. Simultaneous transitions are not possible and their erroneous occurrence (e.g. due to noise spikes) is ignored.

State	A'	A"	B'	B"	UP Cnt.	DOWN Cnt.
0	0	0	0	0	0	0
1	0	0	0	1	0	1
2	0	0	1	0	1	0
3	0	0	1	1	0	0
4	0	1	0	0	1	0
5	0	1	0	1	0	0
6	0	1	1	0	0	0
7	0	1	1	1	0	1
8	1	0	0	0	0	1
9	1	0	0	1	0	0
10	1	0	1	0	0	0
11	1	0	1	1	1	0
12	1	1	0	0	0	0
13	1	1	0	1	1	0
14	1	1	1	0	0	1
15	1	1	1	1	0	0

Table 3.1 Quadrature State Table

These states can be entered into a Karnaugh map and the resulting equations simplified to obtain:

$$\text{Down count} = (\underline{A'B''} + A'B'')\ (A''\underline{B'} + \underline{A''}B')$$

$$\text{Up count} = (\underline{A'}B'' + A'\underline{B''})\ (\underline{A''B'} + A''B')$$

(NOTE: Underlined terms are NOT terms.)

which can be implemented with simple EOR and AND logic. This logic is then transferred directly into schematic form on a computer. The schematic logic diagram can be tested using an available simulation package and in a final step converted automatically into an EPLD 900 programming file.

The EPLD 900 implementation in Appendix A titled "filter" contains the decoder logic and several pre-processing stages. The quadrature decoder is the upper part of the circuit with CHA and CHB quadrature inputs. Looking from left to right the four D flip flops and AND/NAND gates perform a voting filter operation. Immediately following this are two J flip flops which create the ' and " states. Finally the logic for the up/down counter equations is implemented. The up/down clock signals are fed directly to individual channel counters to record encoder pulses.

As part of the quadrature decoder circuitry, additional logic to initialize a user selected counter channel upon the arrival of an index pulse from the robot arm joint encoder was developed. This logic is found on the lower part of the

EPLD 900 schematic diagram. The index pulse is used to calibrate relative encoder counts with respect to absolute potentiometer angle readings. To set up a counter channel to be initialized by an index pulse, the controller program loads the initializing value into the index load register. Next a mask word is written to the unmask register to prepare the index detect circuit to be triggered by an index pulse event. When an index pulse arrives it resets the index detect circuit so that no further pulses are recognized, sends a load pulse to initialize the local encoder pulse counter, and flags the event by setting the corresponding channel status bit of the status register. Using this process the controller program can direct the initialization of any individual encoder counter during joint calibration.

3.3.4.4 System Testing and Integration

The quadrature encoding and VMEbus interfacing circuitry was designed and prototypes were constructed to confirm their operation. Simulations of logic functions for the EPLD's were also run to test logical EPLD operation.

Following this, the board layout was designed. Close spacing of board components was required to fit all logic and power components on the board. To ensure that all components would fit prior to construction, it was necessary to measure the space taken by each component and power connection. The components and connectors were arranged on computer as they would appear when assembled. This process was also used to minimize the length of address, data, clock, and control lines so that induced noise and signal cross-talk was avoided as much as possible.

3.3.5 PUMA Interface Board (PIB)

The PUMA interface board (PIB) connects the VMEbus controller to the Unimate chassis (refer back to Figures 3.1 to 3.3 for an overview). This connection from the VMEbus analog to digital converter, digital to analog converter and encoder boards is made using twisted pair shielded cables to reduce noise effects. Refer to Appendix section A.4.1 for wiring lists. The PIB is located in slot number fifty of the Unimate chassis servo bus. See Figure 3.5 for an graphic overview of the PIB in the Unimate chassis. Signals to and from the existing PUMA servo amplifier and arm sensors are wire wrapped to the backplane of this slot. A complete wiring list for slot fifty backplane connections is found in Appendix section A.4.2.

A PIB protection/isolation strategy was defined and followed. TTL level signals are isolated with TTL inverting buffers or with optocouplers. Optocouplers are used when non-TTL signal levels are found on the Unimate side or when Unimate TTL signals originate from boards which contain

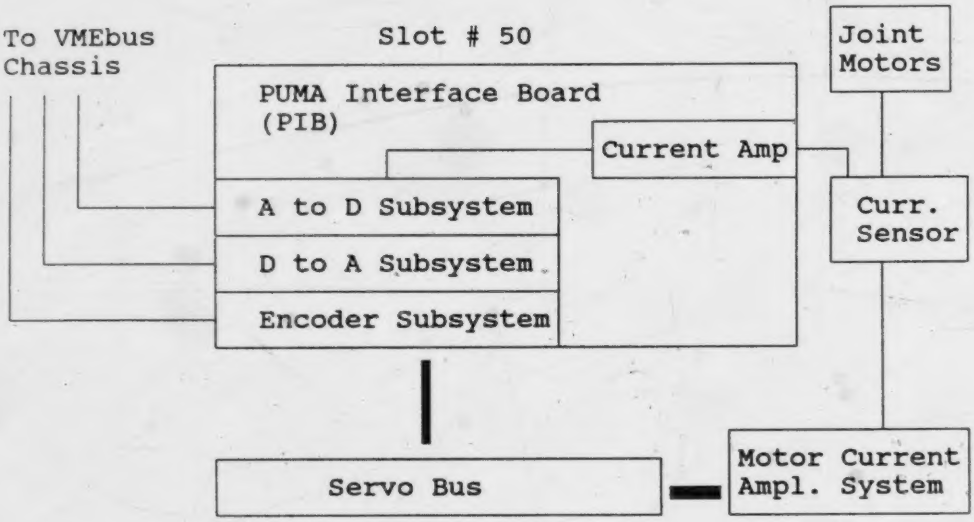

To VMEbus Chassis
Slot # 50
Joint Motors
PUMA Interface Board (PIB)
Current Amp
A to D Subsystem
Curr. Sensor
D to A Subsystem
Encoder Subsystem
Servo Bus
Motor Current Ampl. System

Figure 3.5 PIB in Unimate Chassis

higher voltage circuitry (greater than five volt logic). Analog isolation is implemented with either a voltage follower or a differential amplifier stage. In all cases the desire was to prevent loading of existing monitored circuits and prevent damage of either system during a hardware failure.

The signals sent to the analog to digital converter come from the joint potentiometers, from the system status signals found on the power amplifier control board, and from a current measurement amplifier and separate current sensor board. Joint potentiometer signals are isolated using a voltage follower circuit. The low-pass filter included in this circuit has a 159 Hz three dB point and is used primarily to filter out a large 2.5 MHz noise signal. The system status signals, SS RON HI, STOP L, and BRK RLS MES, indicate that a system fault has occurred (refer to Appendix section A.5.2 for more details on their function). These signals come from the Unimate power amplifier board and hence are optoisolated. Finally a joint motor current signal is supplied by the current amplification/sensor measurement board subsystem (see next section).

The digital to analog converter signals are isolated in two ways. Digital to analog converter signals zero to five send joint servo commands to the Unimate motor current amplifier. All connections to the Unimate servo amplifier are made through servo bus wirewrap backplane connections.

Servo commands coming from the six 6503 Unimate servo boards are switched through one, eight channel dip switch and servo commands coming from the VMEbus controller are switched through another. This arrangement allows for selective servoing of any of the robot joints from the Unimate or VMEbus controllers. The last digital to analog signal from converter channel six is optoisolated before sending it to the power amplifier control board. This line is used to prevent the robot arm power from being turned on prior to VMEbus controller, control algorithm execution.

Finally the quadrature encoder and index pulse encoder signals are sampled using standard Schmitt triggered inverters. These inverters, compatible with the MC14584B CMOS inverters used for Unimate encoder signals, cut down on encoder signal noise. Optoisolator buffering was considered for this interface but it was found to be too expensive and introduced phase distortion at higher frequencies.

Analyzing the PUMA interface, at least on the surface, it appears that simpler buffering techniques might have been used. For instance, analog signals might have been passed directly between the VMEbus and Unimate chassis without operational amplifier buffer stages. Monitoring the power and signal lines at various Unimate chassis locations, however, revealed that periodic noise and power spikes occur especially when the arm is moving. To protect the expensive VMEbus equipment from these transients the interface was made as safe and noise free as possible.

3.3.6 Current Measurement Amplifier/Sensor Board

The current sensor board measures the current flowing from the PUMA servo motor current amplifier to one of the first three robot motors. It is presently configured to measure the motor current of joint two.

The schematic for the amplifier and sensor circuits, labelled PUMATORQ, found in Appendix A, is divided into high and low power sections. The high power section is found on the current measurement board (refer to Figure 3.4) and the low power section on the PIB board. The high power sensor board places a 0.05 ohm equivalent resistor in the current path of the joint motor. This resistor made from two 0.1 ohm, 5 watt power resistors was tested under a five amp load for ten minutes and its 0.0496 ohm value was found to be temperature stable. It is smaller, by a factor of ten, than the current amplifier resistors and thus does not have an appreciable effect on the amplifier operation. The differential voltage across this resistor is sent through a shielded cable to the current measurement amplifier circuit found on the PIB.

The current measurement amplifier circuit found on the PIB consists of a differential input and an amplifier stage. The differential input stage found on the PIB isolates the current sensor board, which measures high power (+/-40 volt,10 ampere) motor currents, from the lower power PIB amplification circuitry. The differential stage also provides longitudinal noise rejection for noise induced on the cable connecting the current sensor board to the PIB current amplifier.

The gain of the current measurement amplifier can be adjusted, using the one thousand ohm variable resistor, such that the output voltage is numerically equivalent to the motor current. This is possible since it can be shown that the output voltage can be made proportional to the differential voltage across the 0.0496 ohm measurement resistor if the two 20K/82K ohm legs have the same voltage divider ratio. This was ensured during sensor construction by adjusting the variable 25K ohm potentiometer. See the "Calibration and Testing of Boards" section for the board calibration method.

4. DESCRIPTION OF PROPOSED CONTROLLER SOFTWARE

4.1 The Software Environment

The robot controller software runs in a loosely/tightly coupled multiple processor environment. The multiple processor system is shown in Figure 4.1. The TUTOR 68000 and DVME 601B 68010 boards are used as the real-time processors of the system, while the HP9000 and SUN 3/160 systems perform non-real-time tasks. The TUTOR 68000 and DVME 601B processor boards are tightly coupled. Located on the same VMEbus, these processors communicate through a shared dual port memory found on the DVME 601B board. The communication between all other system processors is done using loosely coupled Ethernet or RS-232 connections.

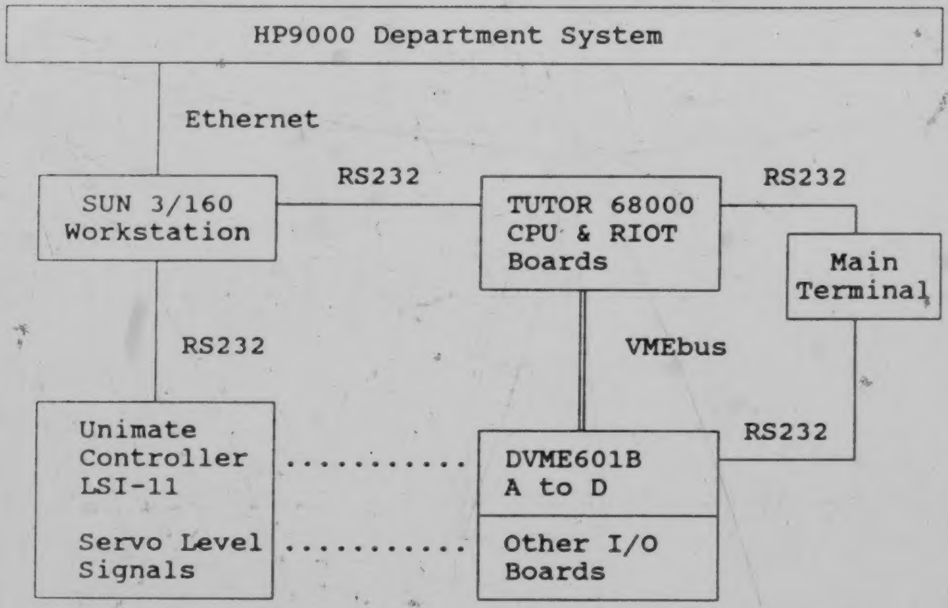
System software tasks are divided into real-time and non-real-time components at the processor level as is done in many typical research robot controllers such as NYMPH and CONDOR. The TUTOR 68000 serves as the main real-time processor. It runs the controller algorithm, user program, coordinates all set up and remote data transfer operations, and supports the user interface which is run from the main terminal. The DVME 601B program pre-processes analog to digital converter data and transfers the results to the TUTOR 68000. The SUN 3/160 and HP9000 systems provide file server and program development facilities. A special communication program runs on the SUN 3/160 which allows the TUTOR 68000 to access the SUN file server. The Unimate LSI-11/73 processor performs the start up initialization of the Unimate controller. It is then disabled and the VMEbus controller takes over the control of the PUMA 560 robot arm.

The programs written for the system are contained in five main files. The files "control.c" and "skeleton.s" contain the "c" language and assembler language code for the controller kernel. The file "user.c" contains a user program which calls higher level subroutines in the controller kernel to perform move operations and can also run a user defined algorithm. The control.c, skeleton.s, and user.c program files are linked together and downloaded into the TUTOR 68000 computer. The user program is executed from the main menu of the controller kernel. The file "sunlink.c" is located on the SUN 3/160 workstation. It is run remotely by the controller kernel found on the TUTOR 68000 computer. Sunlink.c is used to manage RS-232 transfers to and from the SUN 3/160 file system. The file "program601b.s" is downloaded through the TUTOR 68000 to the DVME 601B. This program performs pre-processing of analog to digital conversion data and sends the results to a dual port memory buffer where they are read by the TUTOR 68000.

Program development was done on the HP9000 and SUN 3/160 computers. A cross-compiler/cross-assembler package on the HP9000 computer was used to generate efficient and compact code for the TUTOR 68000 and DVME601B processors. The SUN

3/160 "c" compiler was used to compile the sunlink.c program. Powerful, window based debugging and performance measuring tools are also available on the SUN 3/160.

Program debugging was done using the TUTOR 68000 and DVME 601B monitor programs and developed stop-watch timing routines. Using the monitor programs, breakpoints can be set and the program code selectively executed in between breakpoints or in a single step "trace mode". Hardware registers, processor registers and memory locations can be examined to test for proper program execution. Specially developed, stop-watch timing routines were also developed to time the execution of time critical program segments. These routines, embedded in the source code of the programs to be timed, start a hardware timer before a routine is executed, execute the routine, stop the timer and report the elapsed time. The TUTOR 68000 control algorithm and DVME 601B pre-processing programs were timed using these routines. Based on this information the minimum controller and analog to digital sampling times were estimated.


HP9000 Department System
Ethernet
SUN 3/160 Workstation
RS232
TUTOR 68000 CPU & RIOT Boards
RS232
Main Terminal
RS232
VMEbus
Unimate Controller LSI-11
Servo Level Signals
DVME601B A to D
RS232
Other I/O Boards

—— Processor communication channels

.... Input/Output information channels

Figure 4.1 Connection Of Multiple Processors

4.2 Description of Software

4.2.1 Outline of the Start-Up Process

The software package consists of a distributed set of programs as was alluded to in section 4.1. To begin with, the process that is to be followed to start the system is conceptually outlined. The purpose of this exercise is to explain the roles of the various software modules and the relationship between them. Actual printouts of this process are given in Appendix B section B.1.2.. Main menu functions which are not explained during this phase of the discussion of the start-up process are described at the end of this chapter.

The start-up process begins as the compiled s-record kernel module is transferred from the HP 9000 to the TUTOR 68000 computer. (The HP 9000 is connected by Ethernet to a SUN 3/160. The SUN 3/160 is connected to the TUTOR which in turn is connected to the main system terminal. During the start up process, the main system terminal displays a user interface run by the TUTOR monitor program to the HP 9000) The transfer itself is accomplished using the HP UNIX "cat" and the TUTOR monitor download commands. After completing the kernel transfer, the user exits (logs out from) the HP 9000, but remains logged on to the SUN 3/160 UNIX system. From the TUTOR monitor the user runs the controller kernel. The kernel resides in the TUTOR 68000 memory at location

$1000 (HEX). At this point the assembler code STARTC routine begins executing. This routine saves the TUTOR monitor stack and frame pointer (registers A7 and A6), initializes miscellaneous data structures and "c" language pointers, and finally calls the main() "c" routine. When the user chooses to stop running the controller program, control returns to the STARTC subroutine. The STARTC subroutine then returns control to the TUTOR monitor.

The main() controller routine begins by executing various initialization routines. This is coordinated by the bootup() routine. Under bootup(), the initial() routine is called to prevent arm power from being activated, to set up the address modifier lines for standard addressing, and to initialize the system timer. The timer initialization readies it to generate one millisecond periodic interrupts. These periodic interrupts are used as the base real-time clock for the controller. The routine defset() is then called to initialize keyboard variables, set controller sampling time and process variables, and to define encoder safety limits. The data logging buffer and actively monitored controller variables are then initialized (loginit() and chpick() routines). Following this, pre-multiplication of constant coefficients found in the control law is done to speed execution of the controller algorithm. Finally the control law execution time is checked and displayed for the user's information.

After initialization, the main program menu is displayed. From this menu the user can call routines to calibrate the arm, to test sensors, to move the arm manually, to execute the user program (user.c), or to enter a communication mode. Refer to Figure 4.2 which shows the main menu found in the controller program. At this stage the SUN 3/160 is connected to the remote RS-232 line. The user selects the communication mode from the main menu to access files on the SUN 3/160 system.

The sunlink.c program found on the SUN 3/160 is remotely started using the communication menu's "START" option. To initialize the DVME 601B (68010 board), the 601B local program feature is selected. This feature downloads (using getasc()) the data pre-processing program which is to be run locally by the DVME 601B 68010 processor. The s-records for this program are moved to the 601B dual port memory. After this is completed the "601B command option" is selected. This option using the execon(), dprinitafb(), afbexecut() and prcontafb() routines arranges for the transfer of the s-records to local 601B memory, starts the 601B timer for a one millisecond analog to digital conversion sample rate and starts the 601B pre-processing routine. Under the supervision of the DATEL EXEC monitor program, the pre-processing routine is called every one millisecond after each analog to digital sample. The 601B runs independently,

```
Main Menu:

 ****************Main Controller Menu****************

 Select a number

 1. Redisplay Main Menu
 2. Calibrate Arm
 3. Test Sensors
 4. Manual Mode
 5. User Mode
 6. Change Active Channel (log purposes)
 7. Communication Mode
 8. Exit to TUTOR Monitor Level
 ACTIVE CHANNEL-->x<--
 ==>_


 Communication Mode Menu:

 Select Options
 1. Logfile Download (ENTER SUNLINK FIRST)
 2. 601B local program (ENTER SUNLINK FIRST)
 3. 601B command program
 't' Transparent terminal mode
 'q' for quit
 'x' EXIT SUNlink, reprint menu
 'r' START SUNlink
==>_
```

Figure 4.2 Controller Main & Communications Menus

sampling analog voltages, from this point on. In the last step of 601B initialization, the sunlink program is stopped and the communications routine is exited.

4.2.2 Running the Controller

Returning to the main menu with all initialization done, the VMEbus controller is now ready to control the PUMA 560 robot. To run the manipulator the user must turn on the Unimate controller and obtain the VAL prompt on the SUN 3/160 console. The Unimate controller (see Figure 4.1) is connected to the SUN 3/160 through an RS-232 line on the SUN's "A" serial port. The initialization of the Unimate controller is done by signing onto the SUN 3/160 console and running the puma* program. This program sets up the SUN 3/160 as a simple terminal connected to the Unimate terminal port. Next, the Unimate controller is turned on. Messages will appear asking that VAL II be loaded and the system initialized. "N" should be typed in response to both prompts. The user should now return to the VMEbus controller main terminal.

At the main controller terminal, the manual mode option is selected. Upon selecting the manual mode, the reqarmon() routine is executed. In general, reqarmon() is called when any of the main menu manual mode, calibrate, or user options are selected. The routine starts the control algorithm and prompts the user to depress the Unimate arm power ON button.

Once this button is pressed and the arm power is on, the Unimate LSI-11 processor must be halted by toggling the RUN/Halt toggle switch first to the halt then back to the RUN position. If this is not done and an attempt is made to move the arm from its "power on" position, the LSI-11 supervising program will automatically turn the robot arm power off. This happens because the servo boards are still monitoring the arm position and attempt to keep it from changing by signalling the LSI-11 to turn the arm power off.

Immediately after reqarmon() is executed the manual mode() routine is called. Once in manual mode the user can move any robot joint by 100 or 2000 encoder points at a time using the main terminal keys shown in Figure 4.3.

```
Manual Mode Menu:

********MANUAL MODE EXECUTING*********
   < - Large Move (-)
   > - Large Move (+)
   [ - Small Move (-)
   ] - Small Move (+)
   s - stop robot arm & quit manual mode
   t - tune joint
   v - select which channel to view
```

Figure 4.3 Manual Mode Menu

Joint movement is controlled by selecting a new set point position and automatically ramping the present set point position toward the new desired location. The joint channel being moved is changed by changing the joint channel which is being viewed (calls chpick()). If the user wishes to change the PD controller settings the tune joint option is selected (calls tunepid()). Tuning is done dynamically from the keyboard with the arm power on. Tunepid() also displays a K_i gain since it is intended to test a PID algorithm at a faster sample rate when the controller is implemented on a faster processor than the TUTOR 68000. Presently the manual mode program is set to log the movement of the currently viewed robot joint automatically every time the arm is moved manually.

The control algorithm, updating of controller set points, logging data points to a RAM buffer, and polling for keyboard input mentioned in the manual mode discussion are all implemented by the periodic interrupt routine. The interrupt structure of the controller is set up such that an autovector 7 (IRQ7) interrupt is generated by the TUTOR RIOT card timer every one millisecond. When an interrupt occurs the assembly language HANDLER routine is called. This routine resets the timer interrupt status bit so the next interrupt may be generated. It then calls a "c" interrupt handler (handlerc()) to execute the periodic control, set point, logging and polling functions. The periodic interrupt handler is given highest system priority over all other controller routines. The periodic functions are described further in the following paragraphs.

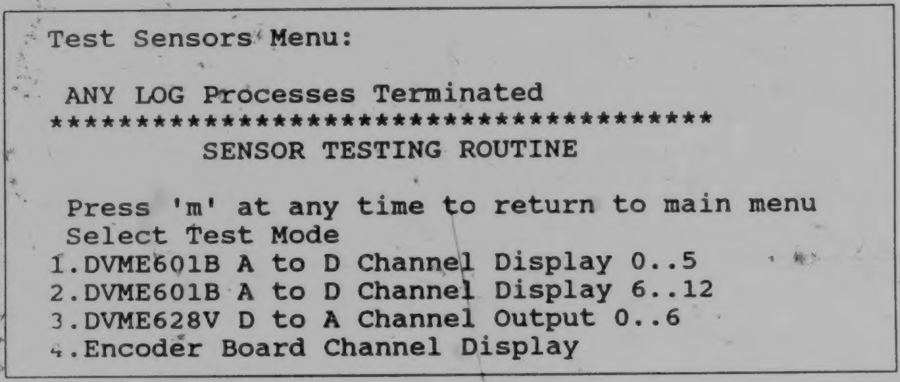
The periodic control function is executed in a regular sampling period. When the control function is executed it sets a control algorithm executing flag, resets the control algorithm interval counter to zero, and then calls the robot controller algorithm. The fixed point PD algorithm (refer to section 4.3 for a more detailed discussion of the algorithm) then begins execution, but it is only partially complete when the next one millisecond interrupt arrives. When the next and successive interrupts arrive the control algorithm interval counter continues to increment. If the control algorithm takes longer to execute than its sampling period then the next attempt to call the control algorithm will discover the control algorithm executing flag is still set. Under this condition the arm power is turned off and the control algorithm stops. This provides a safety mechanism to protect against the piling up of control interrupts which will eventually overwrite program memory and cause potentially dangerous control program failure. Under normal conditions the control algorithm will finish before its sampling interval is complete and the next call to the algorithm will occur normally.

The periodic setpoint updating function found in the interrupt handler is used to ramp the present encoder set point for a given joint towards a new desired set point value. Every trajectory update period the present encoder set point is incremented by "rampstp" encoder points toward the desired new setpoint.

The periodic logging function collects data about a given joint every control algorithm sampling time. The type of controller data to be collected is determined in the loginit() routine which is executed during system initialization. Encoder, setpoint, control servo values and joint current values are presently accessible by the logging function if desired by the user. The logging function can be called from a user.c routine and is presently called automatically from the manual mode routine. Logging operation is controlled using the logon() and logoff() functions. Once a set of data points have been logged, the data can be transferred to a SUN 3/160 file using the logfile download feature of the main menu communication mode. The process takes three to four minutes for a full buffer.

Three main menu options have not yet been described, namely the test sensor, arm calibration, and user modes. The testsensor() routine allows the user to view voltage values on the analog to digital inputs used by the controller, toggles the digital to analog outputs and displays the present encoder count values for each of the joints (see Figure 4.5). Testsensor() is used for calibration and board testing. The robot arm calibration routine sets the joint encoder counter values so that they correspond to an absolute physical position. This is described in greater detail in the next section.

The "user" routine is used by the user to define new robot algorithms and accesses high level controller kernel routines to move the robot arm as desired. Several high level move routines, as a starting point for a larger library such as RCCL [Hayward and Paul,1986], have been programmed for this purpose. The movee(), movea() and moveta() routines move single joints by incremental encoder count angle values or move joints to absolute angles. The vectorje() routine moves all joints simultaneously by an incremental encoder count value.



```
Test Sensors Menu:

 ANY LOG Processes Terminated
***************************************
         SENSOR TESTING ROUTINE

 Press 'm' at any time to return to main menu
 Select Test Mode
1.DVME601B A to D Channel Display 0..5
2.DVME601B A to D Channel Display 6..12
3.DVME628V D to A Channel Output 0..6
4.Encoder Board Channel Display
```

Figure 4.4 Test Sensors Menu

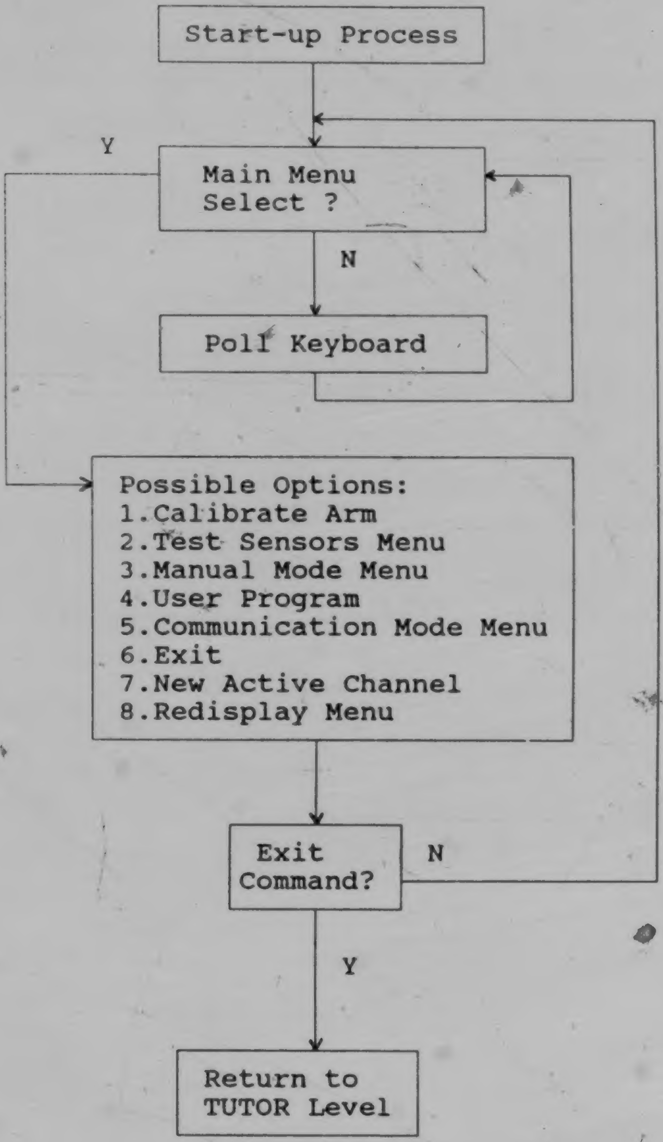

Start-up Process
Main Menu Select ?
Y
N
Poll Keyboard
Possible Options:
1.Calibrate Arm
2.Test Sensors Menu
3.Manual Mode Menu
4.User Program
5.Communication Mode Menu
6.Exit
7.New Active Channel
8.Redisplay Menu
Exit Command?
N
Y
Return to TUTOR Level

Figure 4.5 Menu Structure

4.2.3 Arm Calibration Routine

On the PUMA 560 robot arm there are two devices to detect robot arm joint position. The first device, the potentiometer, can be used to sense absolute joint position but the reading obtained from this measurement is not very accurate. The second method to detect joint position is to use an optical encoder which is much more accurate. Calibration initializes the optical encoder count such that it is relative to some absolute home joint position.

Counter channel initialization during calibration is triggered by encoder index pulses which occur every motor revolution. For the PUMA arm, encoder index pulses occur at intervals of 800 or 1000 encoder pulses depending on which joint is being considered. If the number of encoder pulses to the nearest index pulse from the desired home position is known then the relative encoder count can be related to the absolute angle position.

To calibrate automatically, the less accurate potentiometers can be used to see how many index pulses the arm joint is away from a base index pulse position. For the calibrate() routine this base index pulse position is directly adjacent to the home position which is defined by marks scribed into the robot structure. The potentiometer resolution is not sufficient to measure angular displacements between individual encoder pulses but its reading can be used to identify an arbitrary index pulse found at any joint position. If the potentiometer voltage difference between index pulses is known and the potentiometer value of the home position is also known then an index pulse found at any point in the revolution of the joint can be identified. By counting the number of index pulses the joint is away from the pulse adjacent to the home position it is possible to calculate exactly how far the joint is from home position in terms of encoder pulses. This encoder pulse count value can then be used to initialize the counter channel when the arm joint is exactly at a given index pulse position.

Using the relation between the index pulse and potentiometer reading each individual joint encoder is calibrated to correspond to an absolute joint angle. In the actual algorithm, the robot arm is moved until an index pulse is detected. At this point the arm is moved several encoder points further and the potentiometer reading is taken. The index pulse is then identified and the proper index value for the next index pulse is loaded into the index load register. Following this the next index position is run over by the joint and the encoder is automatically initialized. In the final step the arm joint is moved to the home position to visually confirm that the calibration is successful.

4.3 PD Algorithm

Due to computational limitations of the present controller system it was necessary to select a simple PD control algorithm to compute motor servo commands. A PID algorithm was tried to reduce steady state errors but like the results reported by other researchers [Hsia,et al.,1988] [Leahy and Saridis,1987] [Khosla and Kanade,1986] it was found that the overshoot of such a controller makes it less attractive then the simpler PD controller. The PID controller could not be improved even after careful tuning, for a less oscillatory PID response was obtained at the price of an increased settling time. Plots of controller step responses and further discussion of these results is found in chapter 5.

The PD algorithm is based on the traditional PD controller equation but it has been implemented using fixed point integer arithmetic with constant terms pre-multiplied to ensure a faster sampling interval. The PD equation used is as follows:

control = K_p*errpos + K_d*(errpos-olderr)/delta
errpos = desired set point - actual set point
olderr = error from last sample
delta = sampling interval

The error term is based on desired and actual encoder counts. The delta term is the sampling time of the control algorithm.

In the fixed point program the values of K_p and K_d have been truncated in K_{pf} and $K_{ddeltaf}$ long integer type variables. $K_{ddeltaf}$ is created by pre-multiplying the K_d and 1/delta terms. The PD control law is thus,

$$control = K_{pf}*errpos + K_{ddeltaf}*(errpos-olderr)$$

which can be calculated in fixed point integer arithmetic. The fixed point form has an added advantage. The "control" variable is now in twos complement form which can be sent directly to the digital to analog converter. This converter uses twos complement coding to represent analog voltages.

What errors does such a method introduce in the control law calculation? What is the loss in accuracy because of gains in computing speed? The most obvious difference is that the fixed point controller gains can only be changed in coarser increments than the floating point controller gains. This restricts the tuning accuracy of the fixed point controller. At low K_p gain values (1..50) differences in floating point and fixed point gain representations vary from 100% error to 2% error. For example if K_p=1.9999 then K_{pf}=1 is the closest gain lower than K_p. If K_{pf}=2 causes instability then only K_{pf}=1 can be used. Similarly the K_{df} gain increments differ by 1/(sample_frequency). The other difference between implementations of the controller is that the minimum allowable gains are K_{pf}=1 and K_{df}=1/delta. The gain increment problem is not significant for the range of Kpf gains used (100 and above) and has a slightly larger effect on the selectivity of K_{df} gain increments. The K_{df} gain resolution can be increased by increasing the sample rate. The PD controller implemented in fixed point arithmetic is thus not significantly different from the floating point version and runs at a much higher sampling rate.

5. EXPERIMENTAL RESULTS

In this chapter, results of the experiments that were performed on the PUMA 560 robot using the new controller are presented. Both the tuning and evaluation phases of the experiments are covered. Step response plots and sample trajectory plots are shown. Subsection 5.1.2 discusses a high frequency resonance problem which was observed during the tuning process. In section 5.2 measurements of joint motor currents (joint number two) are presented.

5.1 Tuning the Controller

5.1.1 PD and PID Controller Tuning

In a digital controller the controller sampling rate has a significant effect on the trajectory tracking performance of a given controller [Shin and Cui,1988]. The sampling rate has been shown to directly affect the controller performance of PUMA 560 [Y.Chen,1987] and other types of manipulators [Khosla,1987]. When the sampling rate is increased and the gains are tuned to their maximum possible values, trajectory tracking performance of PD type controllers is improved [Khosla,1987].

In order to maximize the controller performance, evaluations of floating point and fixed point implementations were done to determine the fastest possible sampling rates.

Classical PD and PID controllers were tested on the TUTOR 68000 board running at 12 MHz (no coprocessor) and the following results were obtained:

Algorithm	Execution Speed
PID Floating Point	77.5 msec
PD Floating Point	37 msec
PID Fixed Point	2.7 msec
PD Fixed Point	2 msec

Figure 5.1 Controller Algorithm Performance

Based on these results it was decided to try running PID controller algorithms at a sampling rate of four millisecond. The PD algorithm is currently being run in the present control program at a three millisecond sampling rate.

The tuning process was based on two different methods. The first method used was the Ziegler Nichols approach for mechanical servo systems. This was used as a first guess for the first three PUMA joint controllers. (It was found to be unusable for joints four to six since the predicted gain values turned out to be very low.) Once the initial gain estimate was made, the joints were tuned dynamically using the tuning option found under the controller program manual mode menu. The value of K_p was increased until the step response began to show overshoot. Then K_d was raised to damp the overshoot. This process was continued until maximum gains without overshoot were found. Data for test step response and simultaneous joint movement plots was then taken to confirm controller tuning.

The tuning process is illustrated for joint number one in Figures 5.2 to 5.4. The oscillatory response of joint one is shown in Figure 5.2a for K_p=768. The period of the oscillation (Tp) is sixty milliseconds. Using this value the suggested Ziegler Nichols parameters for a PID controller [Murrill,1967], modified for a fast electromechanical system, were calculated:

$$\begin{aligned} K_p &= 0.6*Kmax, & \qquad K_p' &= 0.5*K_p \\ K_i &= K_p/(Tp/2) & \qquad K_i' &= 0.25*K_i \\ K_d &= K_p*(Tp/8) & \qquad K_d' &= K_d \end{aligned}$$

$$\begin{aligned} K_p' &= 0.5 * 0.6 * 768 = 230.4 \\ K_i' &= (460.8 / (0.03)) * 0.25 = 3840 \\ K_d' &= (460.8 * 0.0075) = 3.45 \end{aligned}$$

The step response resulting from these PID parameters is shown in Figure 5.2b. A four millisecond controller sampling time is used for this step response. The step response shown oscillates about the final set point position and obviously it is not suitable for moving the robot joint reliably without overshoot. After successive attempts at tuning the PID controller a less oscillatory response was obtained. This response still had an overshoot problem and it decayed much more slowly to the final set point value. Similar oscillatory step responses were found for the first three joints.

To avoid overshoot, a PD controller was implemented for all the joints. For joints one to three, the step response was obtained when each of the joints was in the horizontal position so that the effect of maximum gravity and inertial loading could be accounted for. The plots for these cases are shown in Figures 5.4 to 5.9. The gain parameters used are the default values found in the control.c program listing. Additionally, to test for effects of joint controller coupling, all the joints were moved simultaneously along a ramp trajectory. All joints performed similarly. The response shown in Figure 5.10 shows this trajectory for joint two.

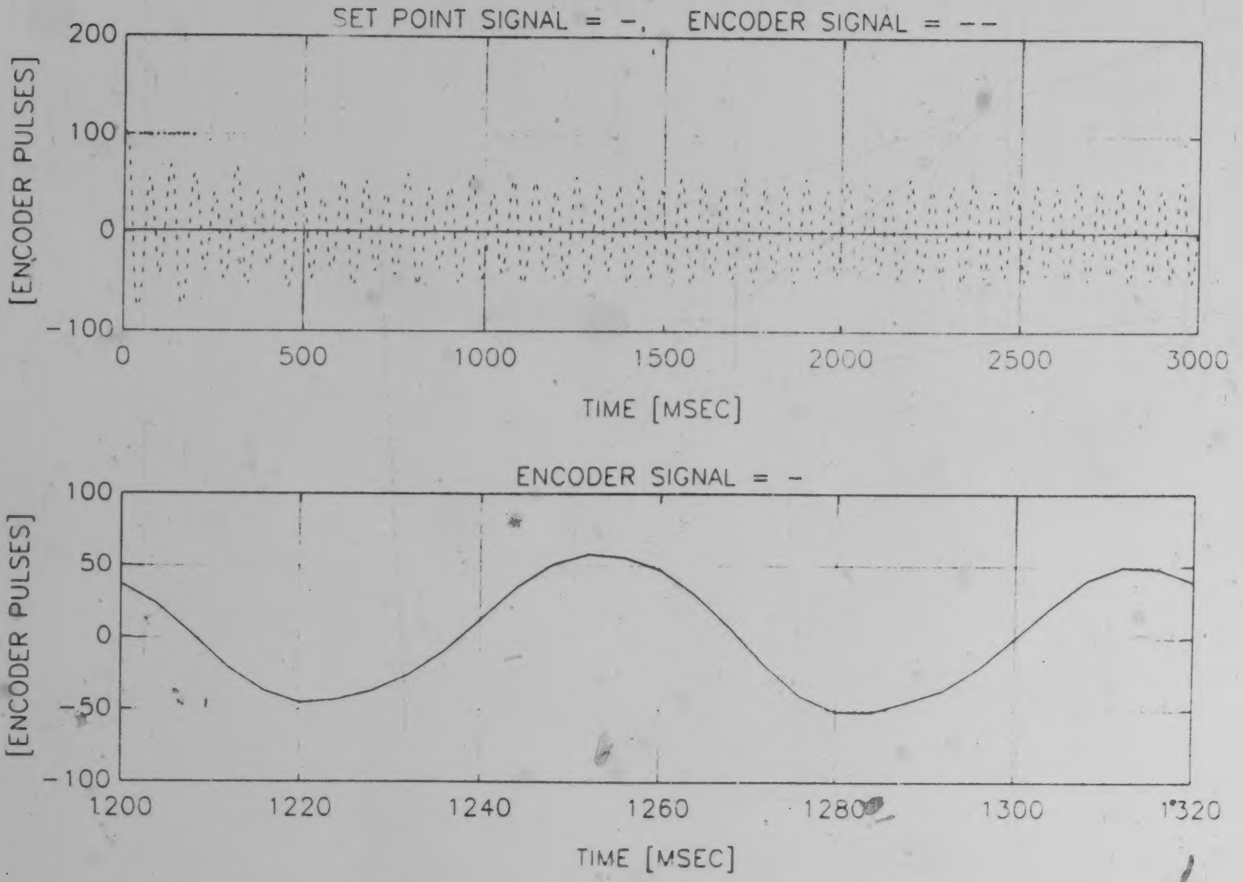


Figure 5.2a Ziegler Nichols Tuning Joint One

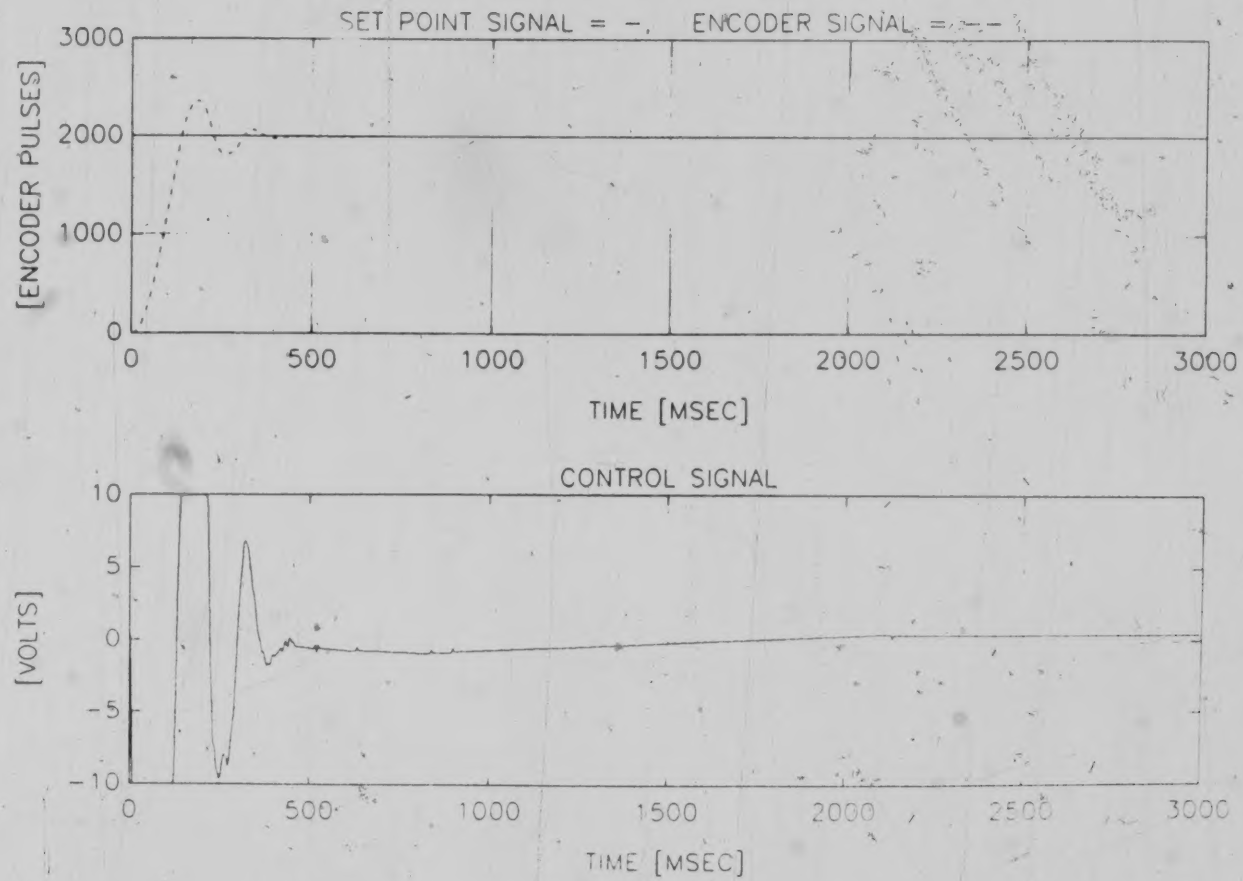


Figure 5.2b PID Controller Based on Nichols Tuning Joint One

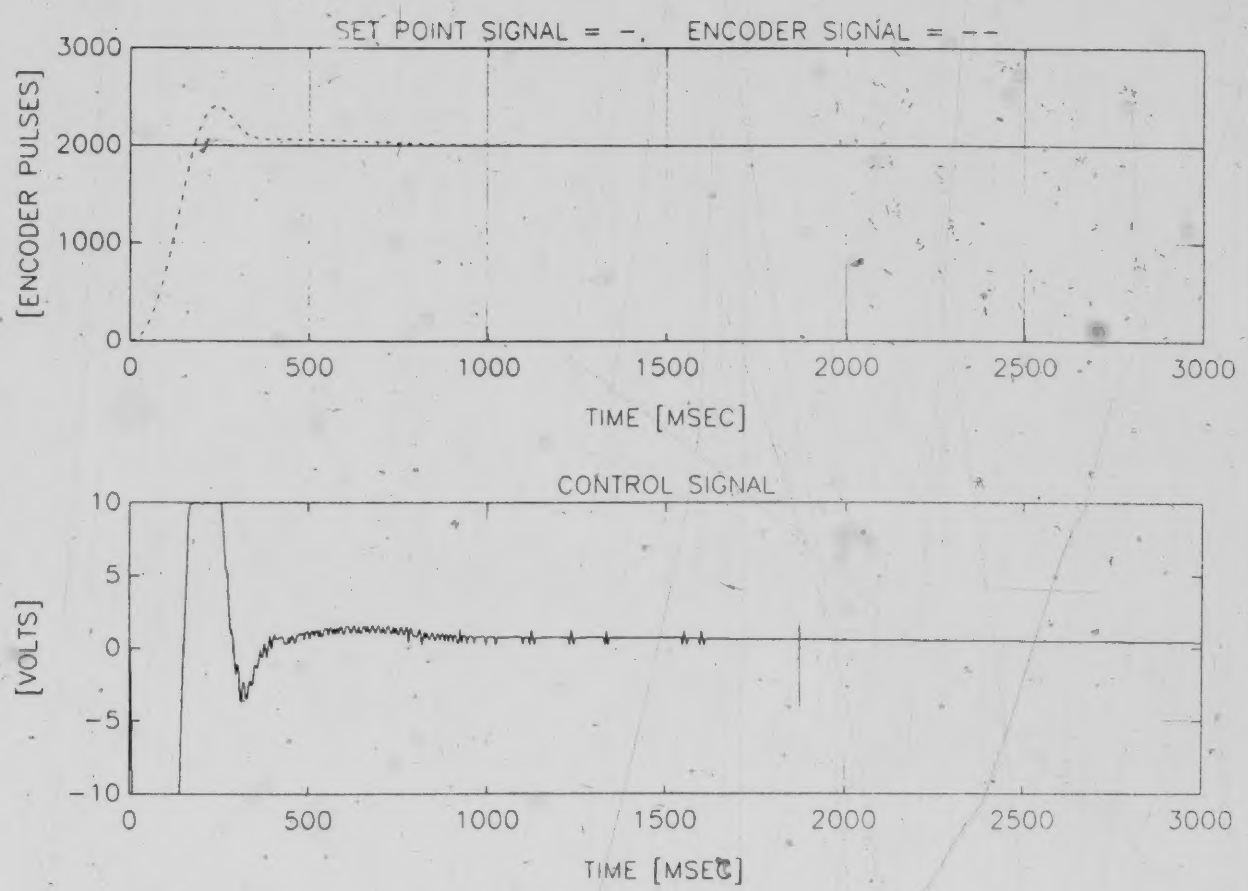


Figure 5.3 PID Controller Improved Tuning Joint One

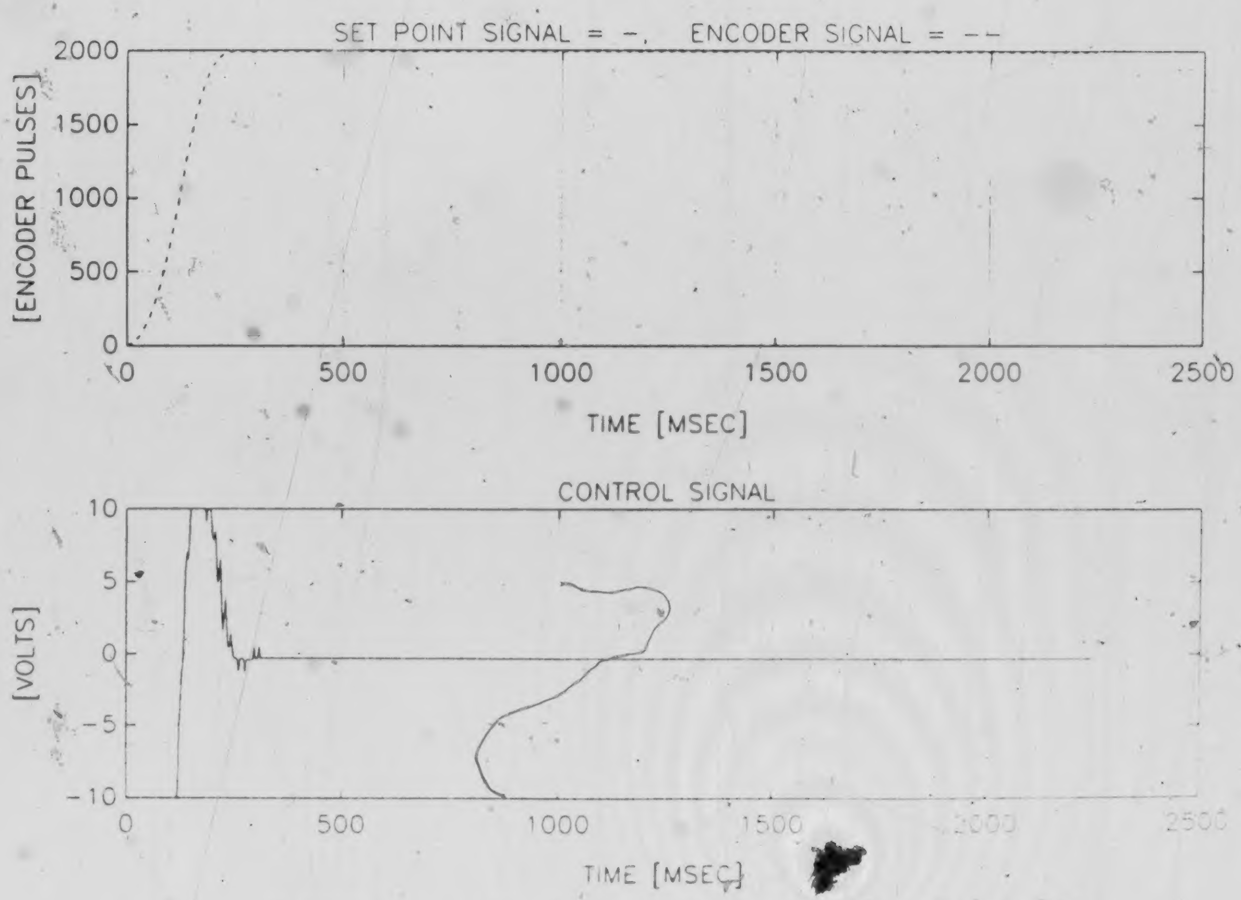


Figure 5.4 Step Response PD Controller Joint One

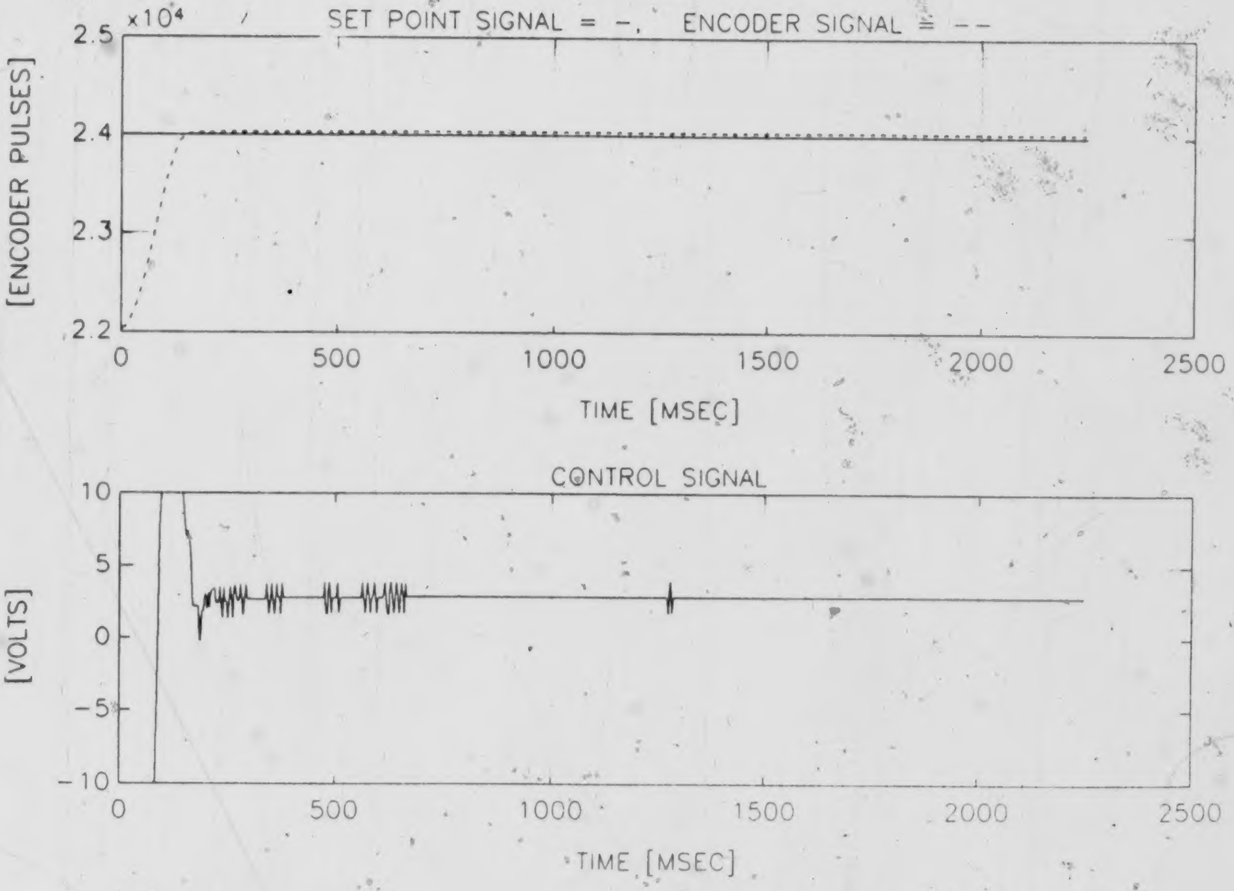


Figure 5.5 Step Response PD Controller Joint Two

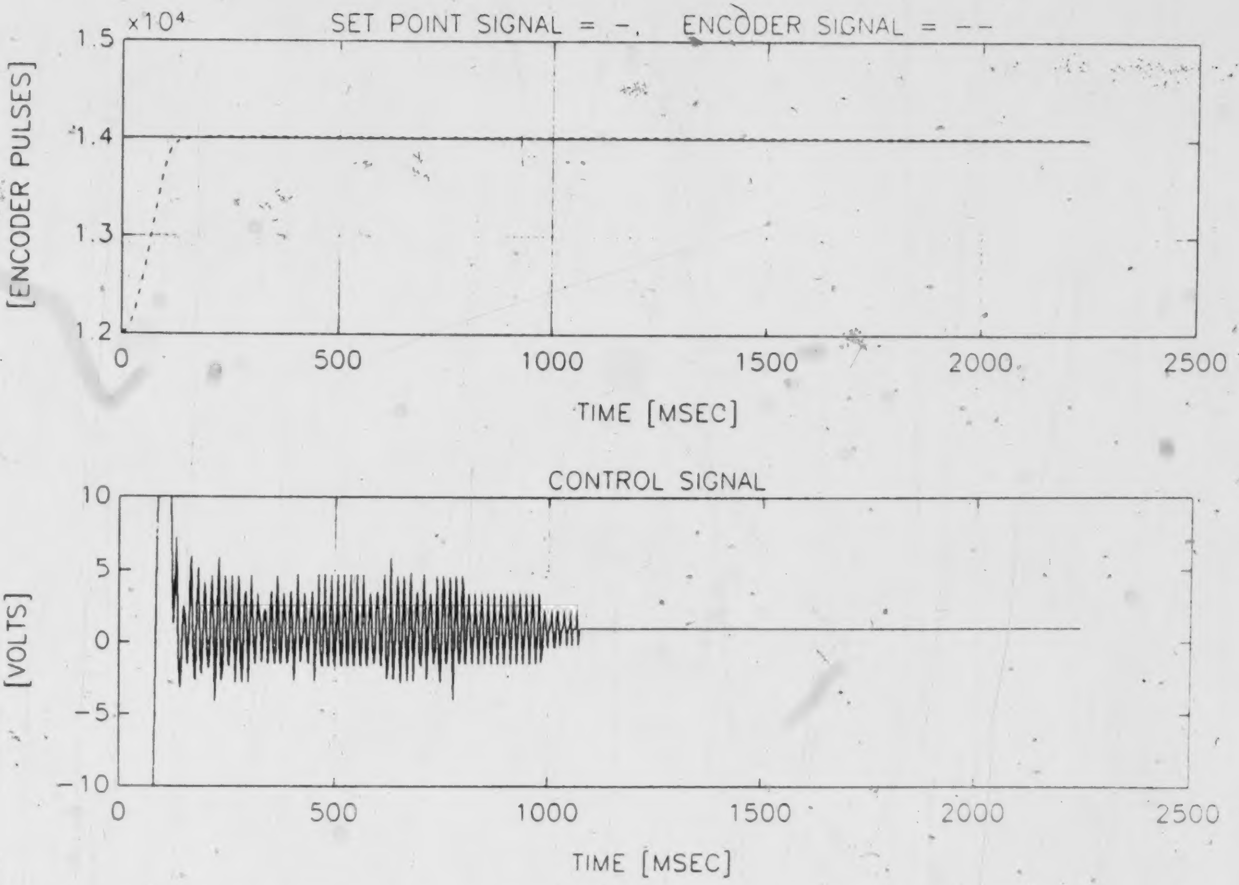


Figure 5.6 Step Response PD Controller Joint Three

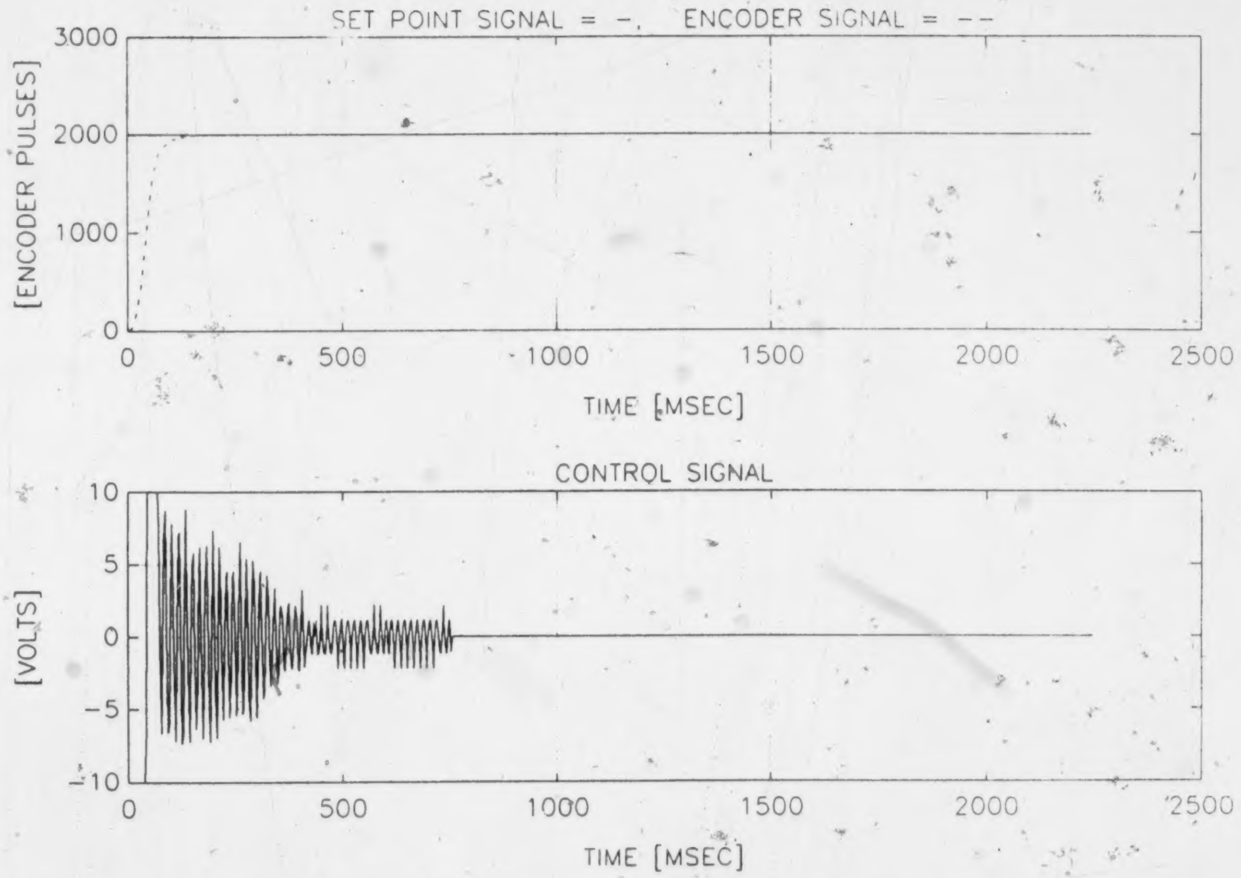


Figure 5.7 Step Response PD Controller Joint Four

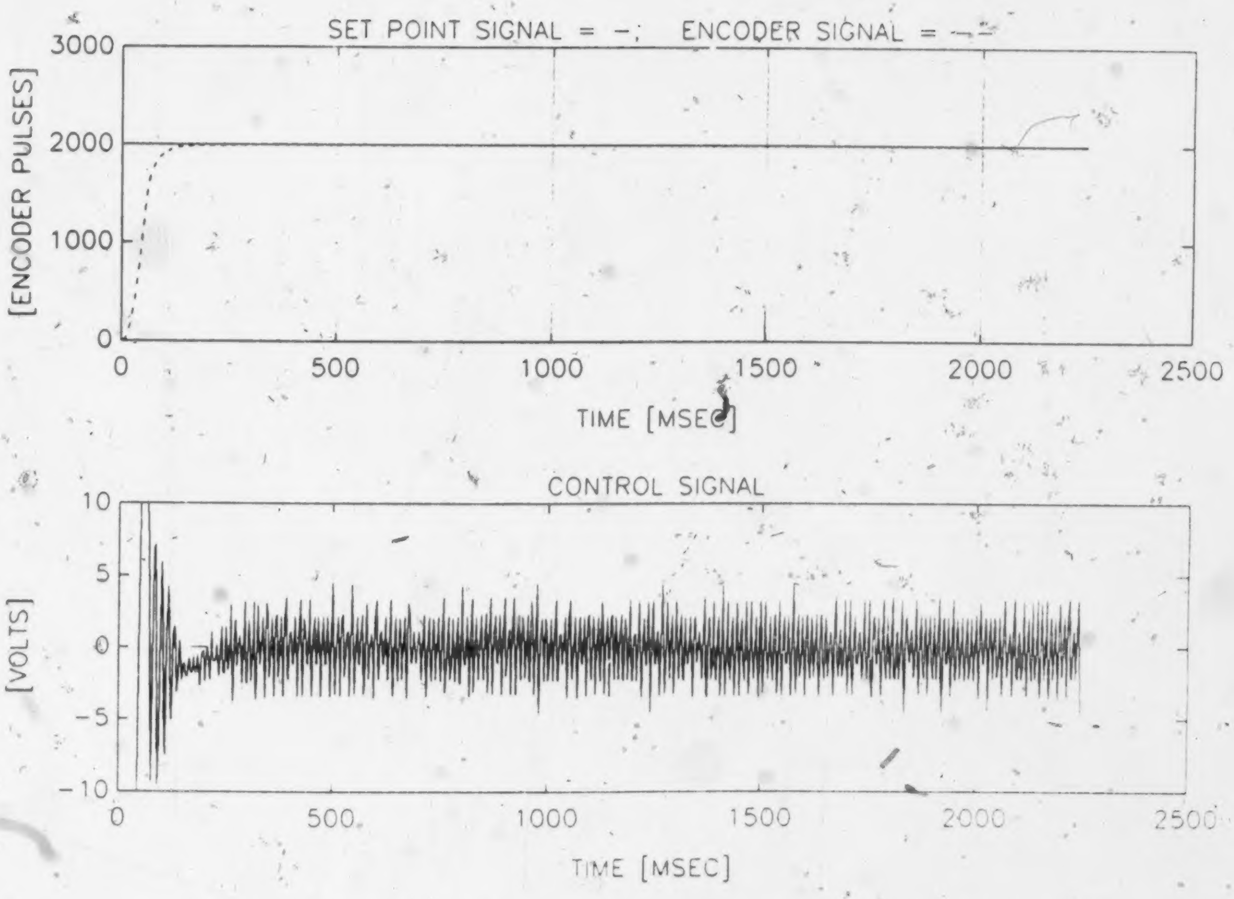


Figure 5.8 Step Response PD Controller Joint Five

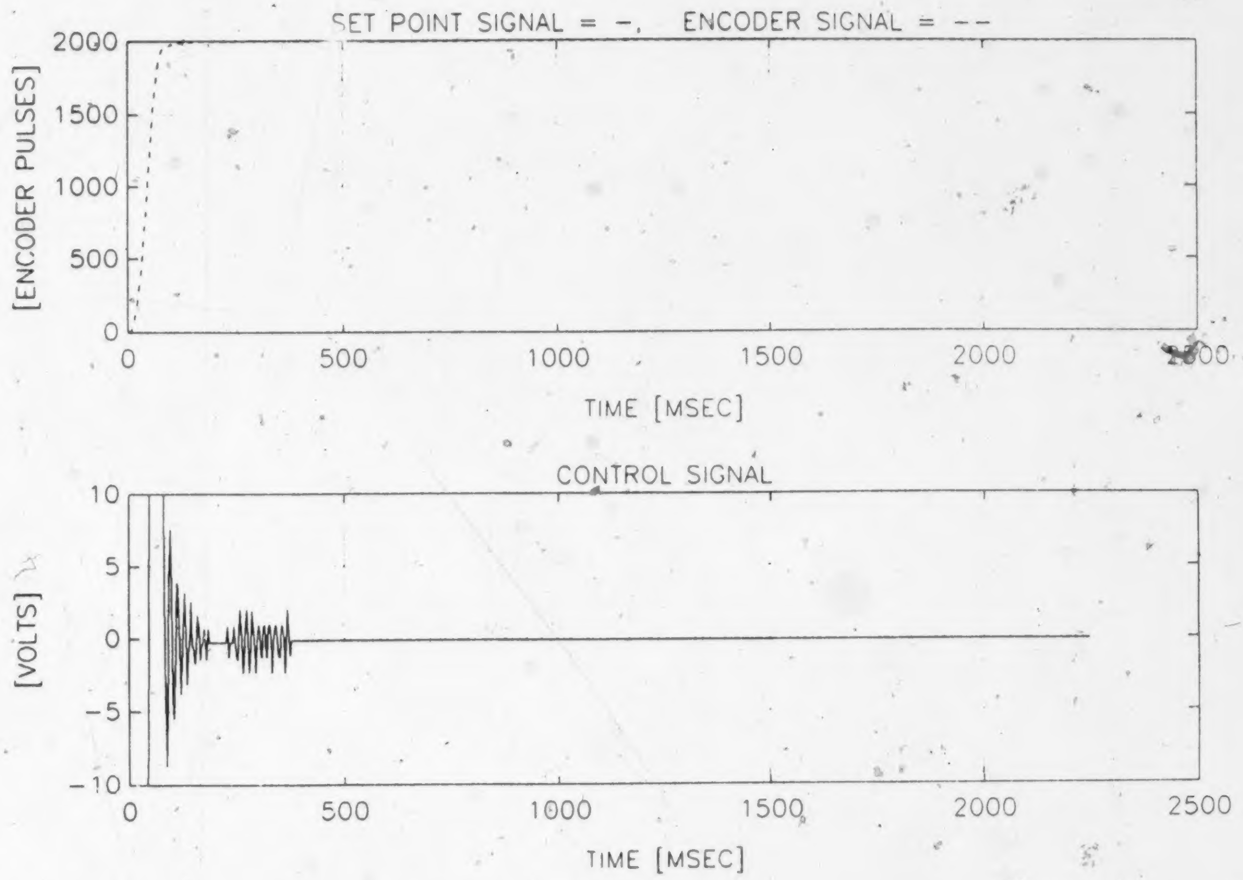


Figure 5.9 Step Response PD Controller Joint Six

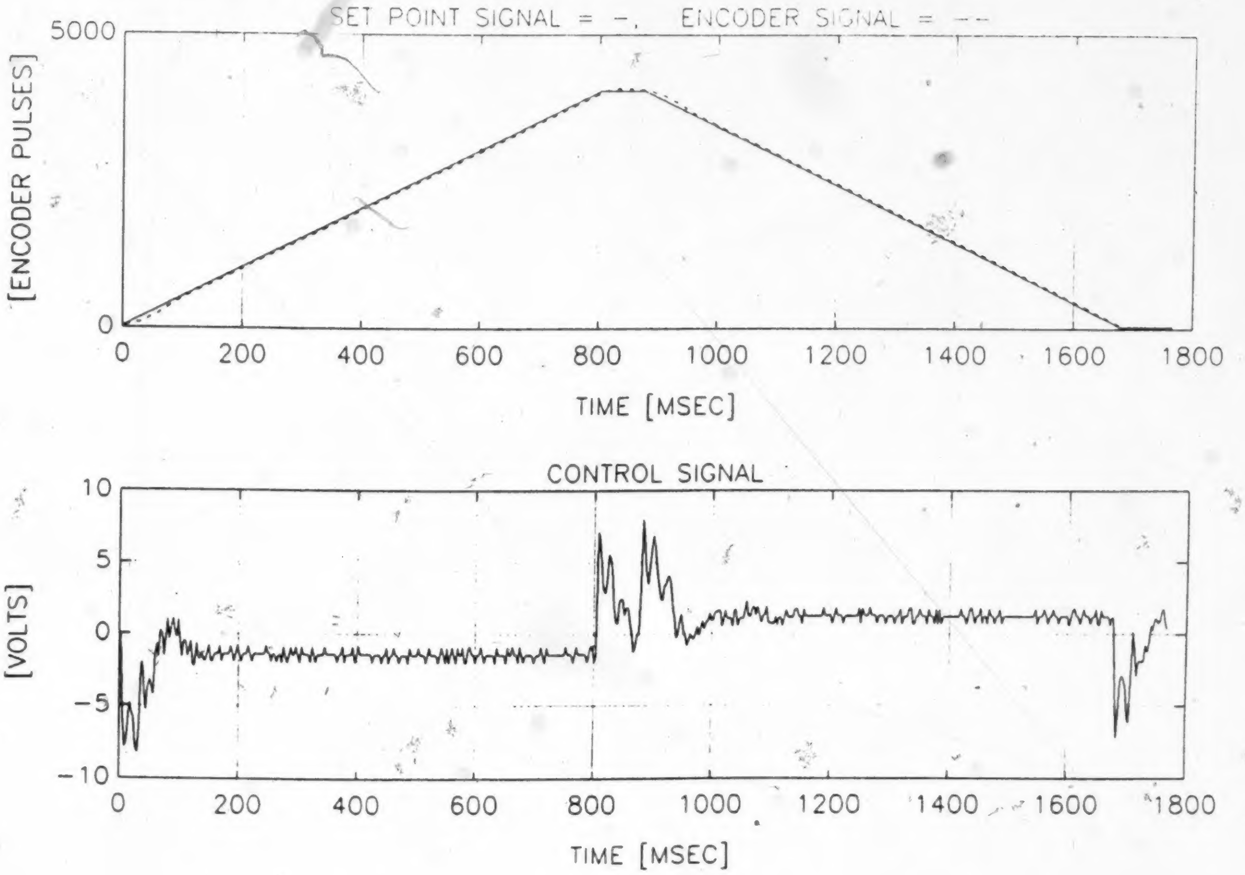


Figure 5.10 Ramp Trajectory Joint Two

5.1.2 Observed "Growling" Phenomenon

The behaviour of the standard digital PD algorithm varies considerably from its continuous counterpart. Specifically, the controller sampling rate has been shown to directly govern the maximum stable controller gains [Y.Chen,1987]. This gain limitation was noticed when the robot joints were tuned. It was found that, when the value of gain K_d was increased to eleven for a four millisecond sampling rate and to fourteen for a three millisecond sampling rate, joint three exhibited a "growling" behaviour. Similar growling conditions were found for the other controller joints as they were servoed through various positions and K_d was increased. This "growling" behaviour was accompanied by a constant, usually small, oscillation of the encoder position about a fixed set point.

The characteristics of this growling were investigated. When the K_d gain was increased to its stability limit or slightly beyond this value the growling did not always begin automatically. If a small 100 encoder point perturbation was applied in manual mode, this growling began and did not stop until the gain was reduced below the stability limit value. These oscillations are shown in Figure 5.11 for joint three with controller gains set at K_p=448 and K_d=14. A power spectral density plot of this signal revealed that the oscillations observed had a main 67 Hz frequency component. This plot is shown in Figure 5.12.

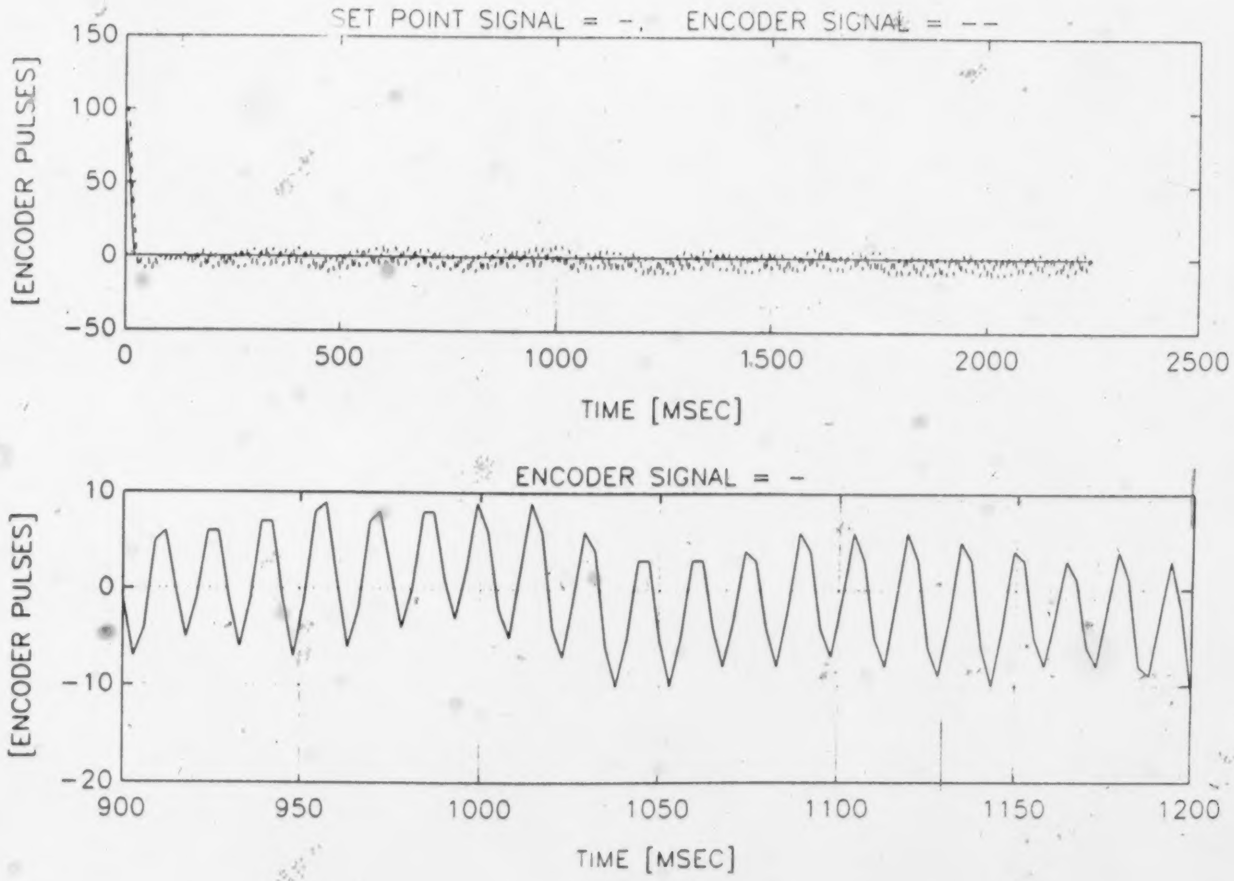


Figure 5.11 Joint Three High Frequency Resonance

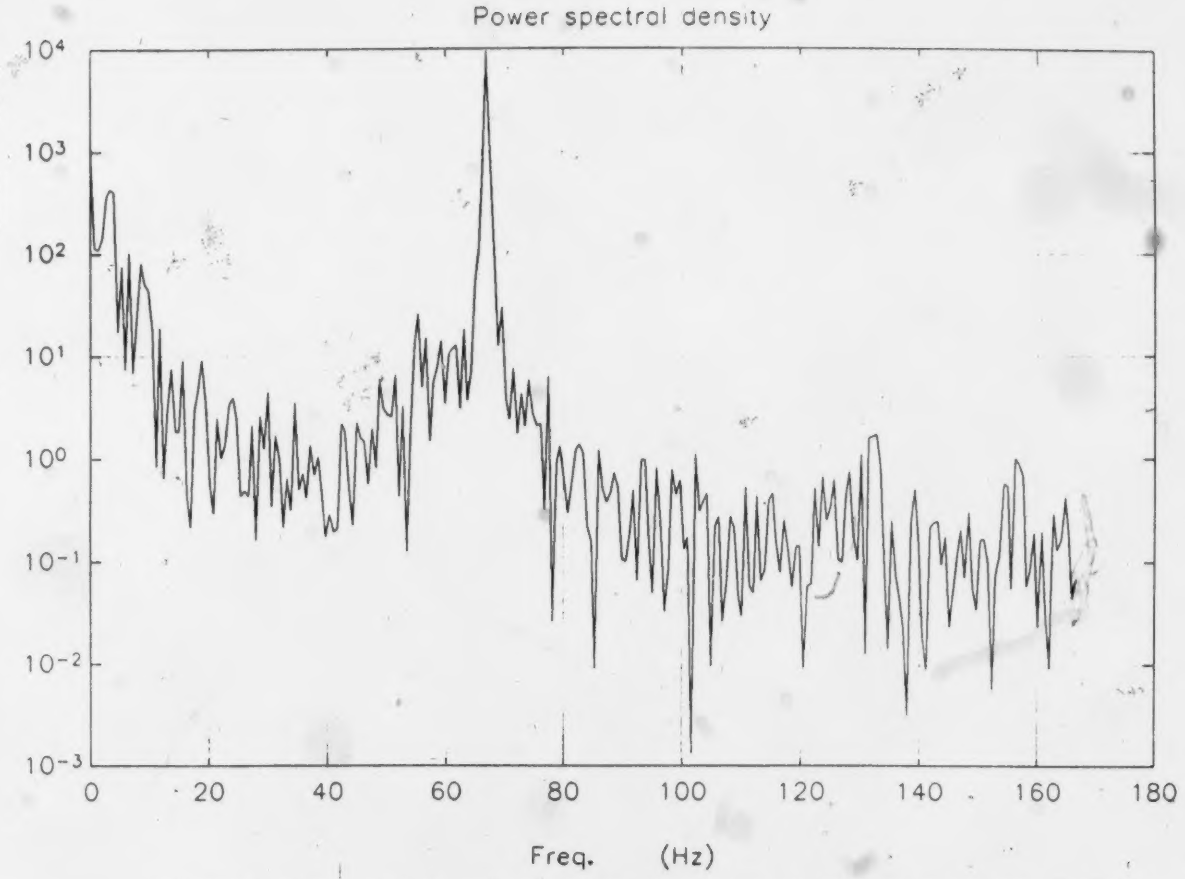


Figure 5.12 Joint Three High Frequency Resonance Spectrum

The origins of the growling behaviour appear to be related to the structure of the digital controller and the robot manipulator. Chen [Y.Chen,1987] derived a third order motor model and experimentally evaluated a PD controller for the PUMA 560. He observed that unlike a continuous system, the digital system grew unstable when the Kd "damping" term was increased and that the maximum stable controller gains depend on the sampling rate. A similar observation was made with respect to the Kd gain for joint three. When the sampling rate was increased, the maximum stable Kd gain could also be increased. Nonlinear friction effects, not accounted for in the PD algorithm, could also be a contributing factor as growling tended to occur most often in the larger, friction-prone joints, one to three. Encoder quantization nonlinearities, causing velocity perturbations at low joint velocities, may additionally contribute to the creation of this phenomenon.

Another possible cause or contributing factor to this growling is the compliance in the joint bearings. It has been reported that vibration modes arising from joint bearing compliance vary according to frequency and damping and depend on the robot configuration [Elosequi,et al.,1990]. It was observed, during test runs, that oscillations tended to damp out quickly when the robot arm was in the vertical position, but became more pronounced and lasted longer when a particular joint was in a more horizontal position.

5.2 Joint Current Measurements

Joint motor current measurements were taken as joint two followed a trajectory. This measurement confirmed the operation of the PIB joint current amplifier and current sensor board. The plots shown in Figures 5.13 to 5.15 show the executed joint trajectory and control and current signals generated for this trajectory. In Figure 5.13 the dotted line represents the current signal and the solid line represents the control signal.

From the current measurement plots we can see that the servo motor current value does not change as rapidly as the control signal. The low pass nature of the amplifier/servo motor system does not allow such rapid variations of the motor current. Further, looking at Figures 5.14 and 5.15, it can be seen that a current signal change corresponding to the control signal command is delayed by about ten milliseconds for this particular case. This delay appears to be primarily due to servo amplifier and motor transient time delays. In Figure 5.15 the joint position movement from its resting position follows the change in motor current.

The current measuring feature of the controller was included to evaluate the use of such a variable for advanced friction compensation and torque based control algorithms. The motor current value used in conjunction with calculated joint velocities can be used to implement various friction compensation algorithms.

Recently a nonlinear system has been announced which uses current measurements to estimate stiction and friction parameters of a PUMA 560 robot [Tarn,et al.,1990]. It is also possible to use the torque constants for the PUMA 560 D.C. motors to estimate the robot arm torque as was done by Armstrong [B.Armstrong,1988] in his robot joint friction research. Many proposed control algorithms use torque variables to control the robot joint motion or to estimate the force applied by the arm. Current measurement is another alternative to using torque measurement devices such as strain gauges and accelerometers.

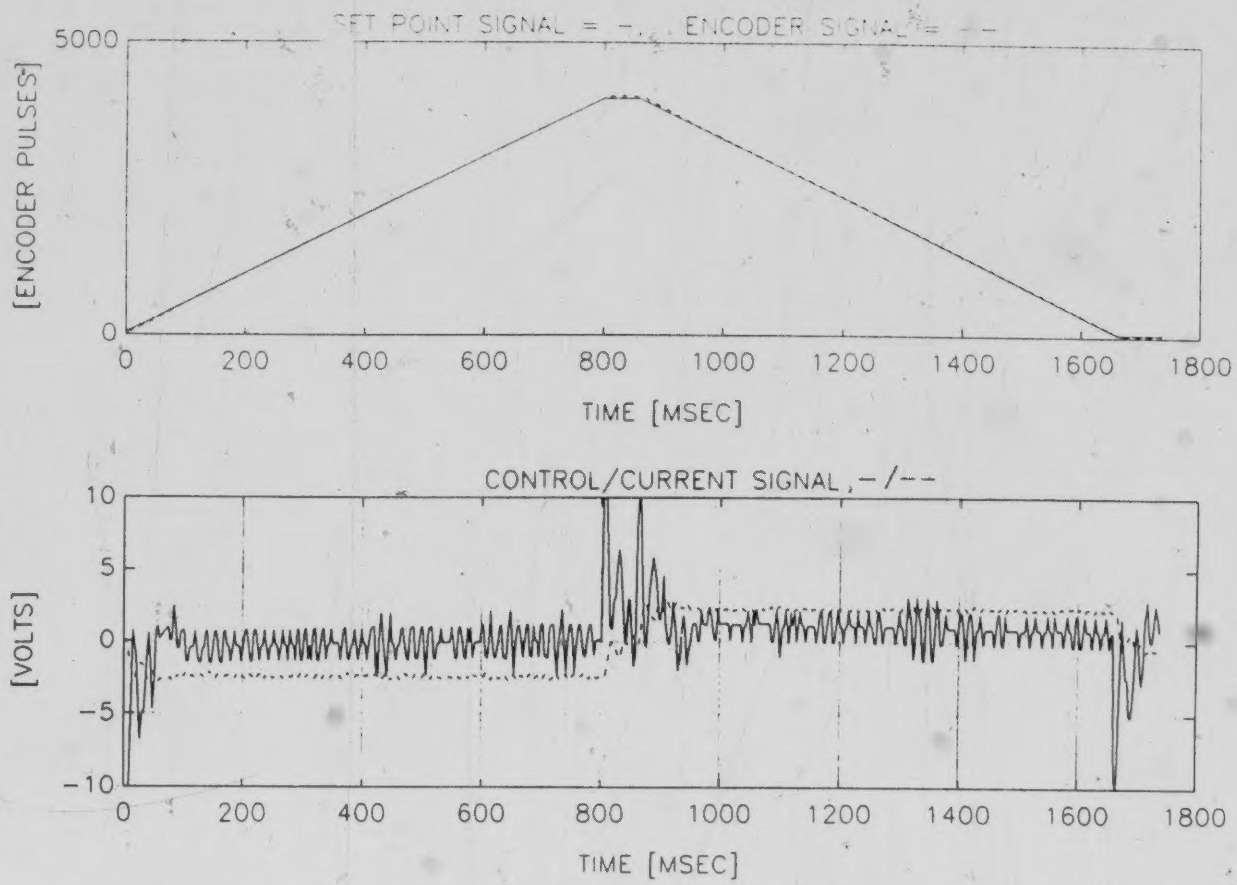


Figure 5.13 Joint Two Ramp Trajectory, Current Measurement

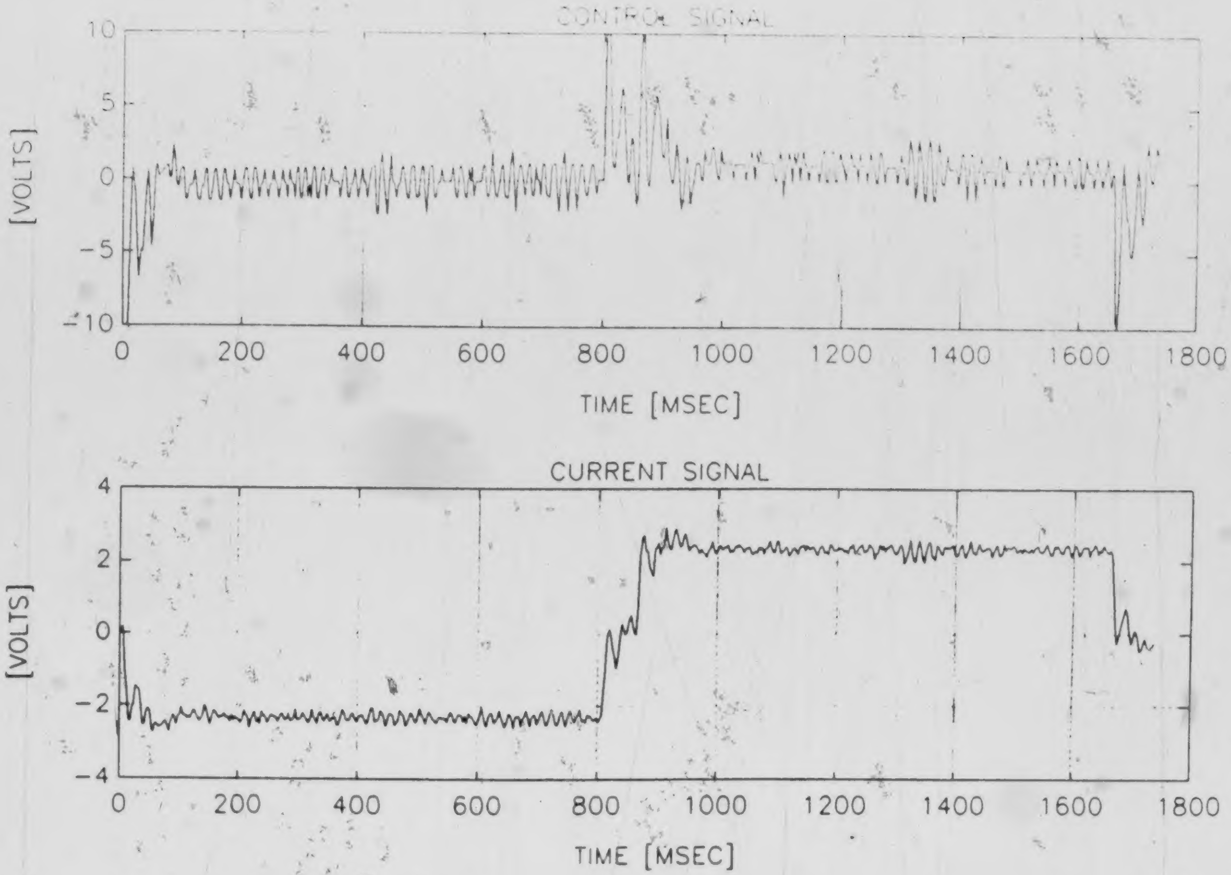


Figure 5.14 Joint Two Control and Current Signals

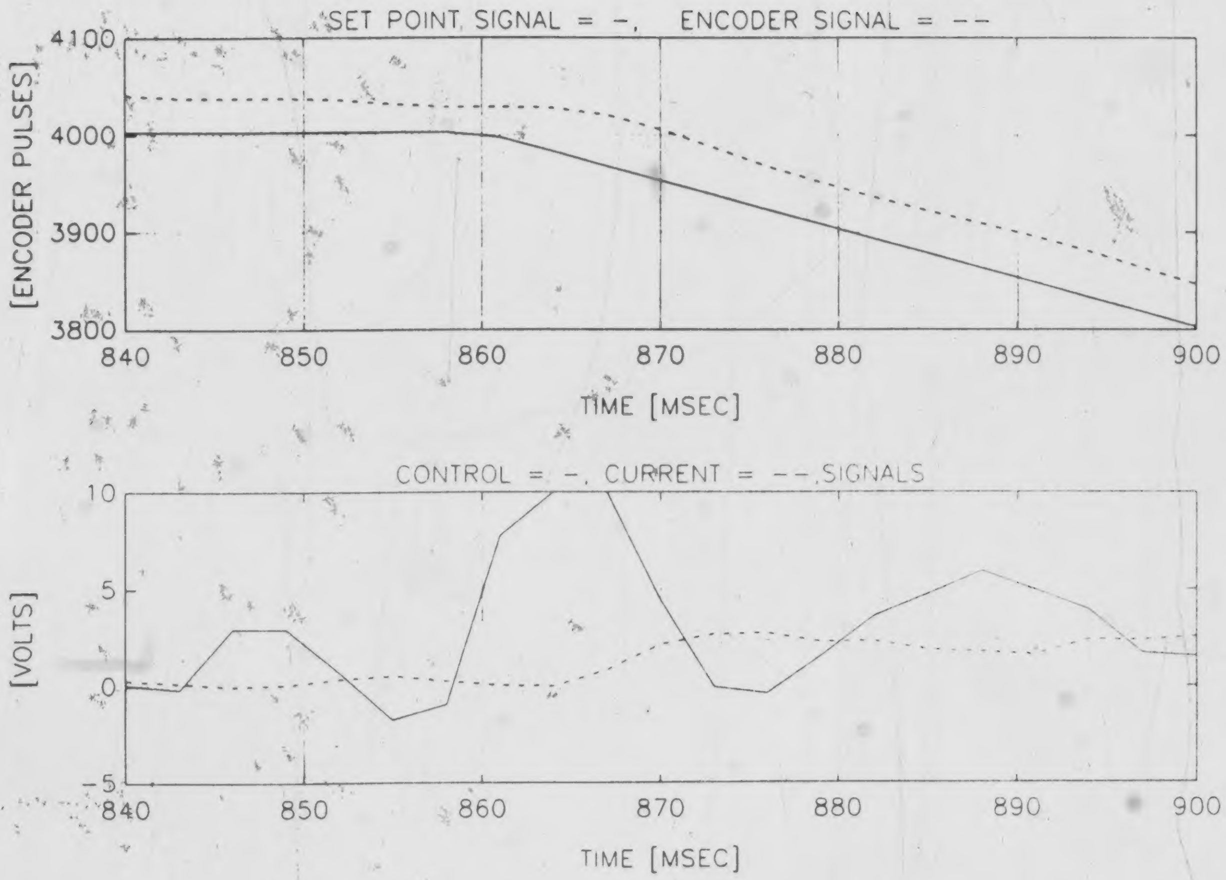


Figure 5.15 Joint Two Ramp Expanded View

6. CONCLUSION & SUGGESTIONS FOR FUTURE RESEARCH

6.1 Conclusion

The VAL II Unimate controller supplied with the PUMA 560 robot was successfully replaced by a VMEbus-based controller. This controller which is similar to the CONDOR and CHIMERA II systems is based on the VMEbus system running 680x0 processors in a homogeneous architecture.

The controller has the following features: (1) additional hardware can be easily added to expand system capabilities, (2) the custom encoder board minimizes the number of clock cycles required to simultaneously collect joint encoder signal information for all joints, (3) the novel integration of servo motor current measurements provides a powerful tool for future control algorithm investigation, and (4) the custom PUMA interface hardware provides a safe interface between the PUMA and VMEbus chassis.

This controller differs from other servo level controllers in several respects. A similar system [Tarn,et al.,1990], in the early development stages, was announced after most of the work on this system was completed. It incorporates current measurement to estimate arm friction parameters. The Tarn controller and other servo level systems differ from this VMEbus controller, in that they are not based on the flexible VMEbus standard and their controller programs are much larger than the kernel of the VMEbus controller (64 Kilobytes).

Detailed system information collected about the VMEbus and Unimate controllers will be a good reference for the addition of further customized hardware. For example, the VMEbus timing diagrams will be useful for the development of new VMEbus based boards. Similarly, the function of various Unimate logic signals has been documented for future error detection purposes.

6.2 Suggestions For Future Research

The present TUTOR 68000 based controller has limited processing, memory, and communication capabilities. It is recommended that a faster coprocessor-equipped central supervising processor be installed to allow implementation of more advanced floating point based control algorithms and user defined algorithms. Like many of the current robot control systems, the addition of parallel processors to share the computational burden is also recommended. More memory is also needed to add these and other features such as an interactive window based user interface and other standard dynamics, kinematics and trajectory planning programs. At the bus level a VME to VME bus adapter to the SUN 3/160 system would be very beneficial as system programs will quickly grow beyond the present 64 KBytes of this controller kernel. Faster transfer of data for program downloading and access to the SUN 3/160 file server is essential with this growth. Finally with the added system memory, program development could be done directly on the SUN workstation. This would avoid the cumbersome HP9000 connection.

The current sensing feature of this controller can be expanded to sample all three main servo motor currents and this new current variable used to investigate more advanced torque based motion or force control algorithms. A printed circuit design for the current board will simplify its construction.

Given the computationally demanding nature of new control algorithms, the capabilities of the VMEbus controller can be stretched by selectively switching joints four to six over to Unimate servo computer control. When the PUMA LSI-11 computer is disabled during initialization the channels still under its control are simply held in their original positions since the PUMA servo board set points can no longer be updated by LSI-11. This allows a VMEbus controller algorithm to process three joints rather than six, speeding its operation.

Additional sensors should be added to the VMEbus chassis to make it flexible for new research. A vision system combined with a force sensor would give it the essential capabilities to interact with its environment more intelligently. Robotics research appears to be moving to more specialized and technically demanding applications

[Fischer,et al.,1990]. Newer systems must not only have computational power but must also have sensors to deal intelligently with their environment.

As more hardware and standard software routines are added to the controller system, the actual controller kernel itself should be further developed and expanded to allow for more flexible coordination of multitasking and multiprocessing system configurations. It might even be adapted to fit into an existing robot operating system environment such as CHIMERA II. Existing libraries such as RCCL [Hayward and Paul,1986] should be integrated where possible.

REFERENCES

Anderson,R.L., "Computer Architectures for Robot Control: A Comparison and a New Processor Delivering 20 Real MFLOPS", IEEE Int. Conf. Robotics and Automation, pp.1162-1167, Scottsdale, Arizona, May, 1989.

Anderson,R.L., "Systems Design for Robot Control With a Scalar Supercomputer", IEEE Int. Conf. Robotics and Automation, pp.1210-1215, Cincinnati, Ohio, May, 1990.

Armstrong,B., "Friction: Experimental Determination, Modeling and Compensation", IEEE Int. Conf. Robotics and Automation, pp.1422-1427, Philadelphia, April, 1988.

Bihn,D.G. and Hsia,T.C., "A Universal Six Joint Robot Controller", IEEE Control Systems Magazine, pp.31-36, Feb. 1988.

Buhler,M., Whitcomb,L.L., Levin,F., and Koditschek,D.E., "A Distributed Message Passing Computational and I/O Engine for Real-Time Motion Control", Proceedings of the 1989 American Control Conference, pp.478-483, Pittsburgh, PA, June, 1989.

Butner,S., Wang,Y., Mangaser,A., and Jordan,S., "Design and Simulation of RIPS: An Advanced Robot Control System", IEEE Int. Conf. Robotics and Automation, pp.470-474, Philadelphia, April, 1988.

Chen,J.B., Fearing,R.S., Armstrong,B.S., and Burdick,J.W., "NYMPH: A Multiprocessor for Manipulation Applications", IEEE Int. Conf. Robotics and Automation, pp.1731-1736, San Francisco, April, 1986.

Chen,Y., "Frequency Response of Discrete-Time Robot Systems - Limitations of PD Controllers and Improvements by Lag-Lead Compensation", IEEE Int. Conf. Robotics and Automation, pp.464-472, Philadelphia, April, 1987.

Clark,D., "HIC: An Operating System for Hierarchies of Servo Loops", IEEE Int. Conf. Robotics and Automation, pp.1004-1009, Scottsdale, Arizona, May, 1989.

Davis,S.H., "Easier Programming With Real-Time Systems", Machine Design, Vol.58, pp.50-55,1986.

Dupourque,V., Guiot,H., and Ishacian,O., "Towards Multi-Processor and Multi-Robot Controllers", IEEE Int. Conf. Robotics and Automation, pp.864-870, San Francisco, April, 1986.

Elosegui,P., Daniel,R.W., and Sharkey,P.M., "Joint Servoing for Robust Manipulator Force Control", IEEE Int. Conf. Robotics and Automation, pp.246-251, Cincinnati, Ohio, May, 1990.

Fischer,P., Daniel,R., and Siva,K.V., "Specification and Design of Input Devices for Teleoperation", IEEE Int. Conf. Robotics and Automation, pp.540-545, Cincinnati, Ohio, May, 1990.

Fu, K.S., Gonzalez,R.C., and Lee,C.S.G, ROBOTICS: Control, Sensing, Vision, and Intelligence, Chp.5, pp.201-266, McGraw-Hill Book Company, 1987.

Goldenberg,A.A. and Chan,L.,"An Approach to Real-Time Control of Robots in Task Space. Application to Control of PUMA 560 Without VAL-II", IEEE Transactions of Industrial Electronics, Vol.35, No.2, pp.231-238, May 1988.

Groover,M.P., Weiss,M., Nagel,R.N., and Odrey,N.G, Industrial Robotics Technology, Programming, and Applications, McGraw-Hill Book Company, 1986.

Hanselmann,H., "Implementation of Digital Controllers-A Survey", Automatica, Vol.23, No.1, pp.7-32, 1987.

Hayati,S. and Venkataraman,S.T., "Design and Implementation of a Robot Control System with Traded and Shared Control Capability", IEEE Int. Conf. Robotics and Automation, pp.1310-1315, Scottsdale, Arizona, May, 1989.

Hayward V. and Paul,R.P., "Robot Manipulator Control Under UNIX: RCCL, A Robot Control C Library", Intl.J.Robotics Res., vol.5, no.4, pp.94-111, Winter, 1986.

Horowitz,P. and Winfield,H., The Art of Electronics: Second Edition, Cambridge University Press, 1989.

Hsia,T.C., Lasky,T.A., and Guo,Z.Y., "Robust Independent Robot Joint Control: Design and Experimentation", IEEE Int. Conf. Robotics and Automation, pp.1329-1334, Feb. 1988.

Ish-Shalom,J. and Kazanzides,P.,"SPARTA: Multiple Signal Processors for High-Performance Robotic Control", IEEE Int. Conf. Robotics and Automation, pp.284-290, Philadelphia, April, 1988.

Joerg,W., CMPE 601: Real-Time Systems Notes, February, 1990.

Kazanzides,P., Wasti,H., and Wolovich,W.A., "A Multiprocessor System for Real-Time Robotic Control: Design and Applications", IEEE Int. Conf. Robotics and Automation, pp.1903-1908, Philadelphia, April, 1987.

Kernighan,B.W. and Plauger,P.J., The Elements of Programming Style, Bell Telephone Laboratories, 1974.

Khosla,P.K. and Kanade,T., "Real-Time Implementation and Evaluation of Model-Based Controls on CMU DD Arm II", IEEE Int. Conf. Robotics and Automation, pp.1546-1555, San Francisco, April, 1986.

Khosla,P.K., "Choosing Sampling Rates For Robot Control", IEEE Int. Conf. Robotics and Automation, pp.169-174, Philadelphia, April, 1987.

Kossman,D., and Malowany,A., "A Multi-Processor Robot Control System for RCCL under iRMX", IEEE Int. Conf. Robotics and Automation, pp.1298-1306, Philadelphia, April, 1987.

Leahy,M.B. and Saridis,G.N., "The RAL Hierarchical Control System", IEEE Int. Conf. Robotics and Automation, pp.407-411, San Francisco, April, 1986.

Lee,I., King,R., and Yun,X., "A Real-Time Kernel for Distributed Multi-Robot Systems", Proceedings of the American Control Conference, Vol.2, pp.1083-1088, 1988.

Leung,S.S, and Shanblatt,M.A., "Computer Architecture Design for Robotics", IEEE Int. Conf. Robotics and Automation, pp.453-460, Philadelphia, April, 1988.

Mack,B. and Bayoumi,M.M., "Design and Integration of New Software for The Robot Controller Test Station", IEEE Int. Conf. Robotics and Automation, pp.866-873, Scottsdale, Arizona, May, 1989.

Mangaser,A.A., Wang,Y., and Butner,S.E., "Concurrent Programming Support for a Multi-Manipulator Experiment on RIPS", IEEE Int. Conf. Robotics and Automation, pp.853-859, Scottsdale, Arizona, May, 1989.

Murrill,P.W., Automatic Control of Processes, International Text Book Company, Scranton, Pennsylvania, 1967.

Naghdy,F., Wai,C.K., and Naghdy,G., "Multiprocessing Control of Robotic Systems", IEEE Int. Conf. Robotics and Automation, pp.975-977, Philadelphia, April, 1988.

Narasimhan,S., Siegel,D.M., and Hollerbach,J.M., "Condor: A Revised Architecture for Controlling the Utah-MIT Hand", IEEE Int. Conf. Robotics and Automation, pp.446-449, Philadelphia, April, 1988.

Narasimhan,S., Siegel,D.M., and Hollerbach,J.M., "Condor: A Coarse-grained Parallel Architecture for Robot Control", Proceedings of the 1989 American Control Conference, pp.484-488, Pittsburgh, PA, June, 1989.

Paul,R.P, and Zhang,H., "Design of a Robot Force/Motion Server", IEEE Int. Conf. Robotics and Automation, pp.1878-1883, San Francisco, April, 1986.

Salkind,L., "The SAGE Operating System", IEEE Int. Conf. Robotics and Automation, pp.860-865, Scottsdale, Arizona, May, 1989.

Schmitz,D., Khosla,P., Hoffman,R., and Kanade,T, "CHIMERA: A Real-Time Programming Environment For Manipulator Control", IEEE Int. Conf. Robotics and Automation, pp.846-852, Scottsdale, Arizona, May 1989.

Shin,K.G. and Cui,X., "Effects of Computing Time Delay on Real-Time Control Systems", Proceedings of the American Control Conference, Vol.2, pp.1071-1076, 1988.

Small,C.H., "Real-Time Operating Systems", EDN, pp.115-138, January 7, 1988.

Stewart,D.B, Schmitz,D.E, and Khosla,P.K., "Implementing Real-Time Robotic Systems Using CHIMERA II", IEEE Int. Conf. Robotics and Automation, pp.598-603, Cincinnati, Ohio, May, 1990.

Takanashi,N., Ikeda,T., and Tagawa,N., "A High-Sample-Rate Robot Control System Using A DSP Based Numerical Calculation Engine", IEEE Int. Conf. Robotics and Automation, pp.1168-1173, Scottsdale, Arizona, May, 1989.

Tarn,T.J., Bejczy,A.K., Ganguly,S., and Li,Z., "Nonlinear Feedback Method of Robot Control: A Preliminary Experimental Study", IEEE Int. Conf. Robotics and Automation, pp.2052-2057, Cincinnati, Ohio, May, 1990.

Whitcomb,L.L., and Koditschek,D.E., "Robot Control in a Message Passing Environment: Theoretical Questions and Preliminary Experiments", IEEE Int. Conf. Robotics and Automation, pp.1198-1203, Cincinnati, Ohio, May, 1990.

500 Series Electrical Drawing Set for VAL II and VAL Plus Operating Systems: 394AC1, Unimation, July, 1985.

500 Series Equipment Manual for VAL II and VAL PLUS Operating Systems 398U1, Unimation, August, 1985.

User's Guide to VAL II, Version 2.0 398AF1, Unimation, February, 1986.

VMEbus Specification Manual: MVMEBS/D1, VMEbus Manufacturers Group, Rev.B, August, 1982.

APPENDIX A - HARDWARE NOTES

A.1 System Level Specifications

A.1.1 System Memory Map

Long Addressing Map:

$FFFFFF-$FF0000 - TUTOR CPU ROM
$FA0000-$FAFFFF - DVME601B A/D Board
$EFFFFF-$EFFC00 - RIOT Duart Channels
$EFFBFF-$EFF800 - RIOT Timer
$EDFFFF-$ED0000 - AM Line Decoder Board
$02FFFF-$020000 - Encoder Board
$00FFFF-$000000 - TUTOR CPU RAM

Short Addressing Map:

$FFC900-$FFC9FF - DVME628V D/A Board

A.1.2 VMEbus Slot Assignment

SLOT #	Description	Jumpering BG3 IN/OUT	Jumpering IACKIN/OUT
1	68000 Tutor Card	N/A	N/A
2	68000 RIOT Card	YES	NO
3		YES	YES
4	AMline Decoder	YES	YES
5		YES	YES
6		YES	YES
7	DVME601B	NO	NO
8	DVME628V	NO	NO
9		YES	YES
10		NO	NO
11		NO	NO
12		NO	NO
13		NO	NO
14		NO	NO
15		NO	NO
16	Encoder Board	NO	NO
17..21		NO	NO

Table A.1 VMEbus Board Assignment

A.1.3 PUMA/Unimate Servo Bus

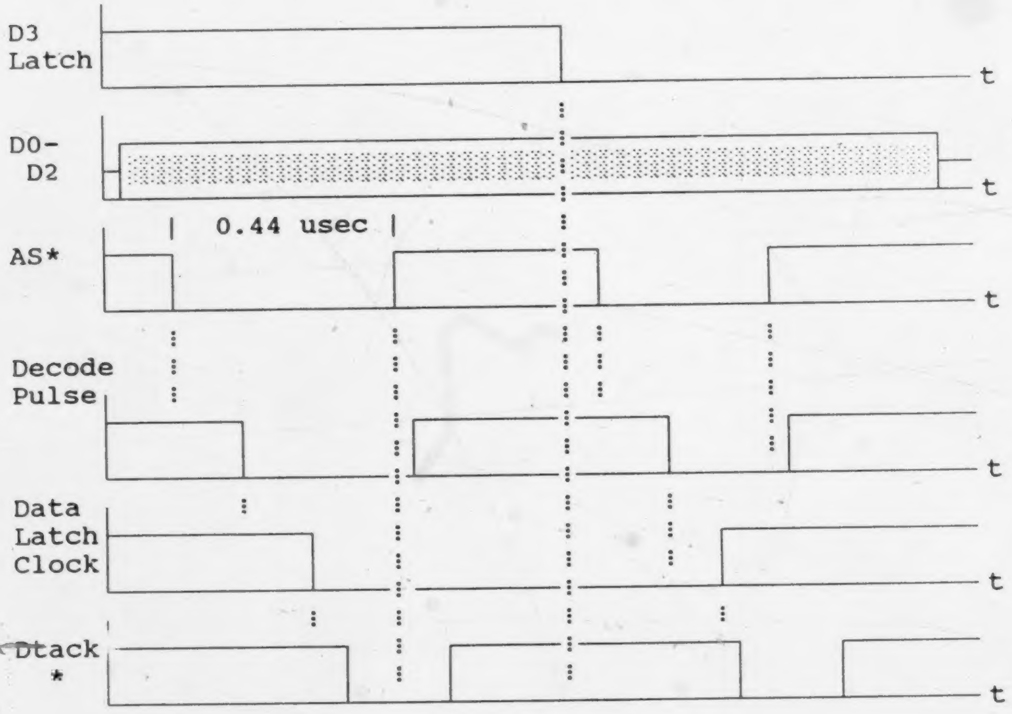
Slot #	Description
J39	B Interface Board
J40-46	Joint Servos 1..7
J47-49	305-1350, REV-D
J50-52	Free, Slot #50 Used For PIB
J53-54	Wired, Use Uncertain
J55	PUMA 760 Slot
J56	Power Amplifier Control
J57	Free, Unused
J58	Cable Card for Robot Arm

Table A.2 PUMA Servo Bus Assignment

A.2 Specifications for Project Boards

A.2.1 AMline Decoder Board - VMEbus

Timing Specification: (12MHz Clock)


D3
Latch
D0-
D2
0.44 usec
AS*
Decode
Pulse
Data
Latch
Clock
Dtack
*
t

▨ Don't Care States

Note: Timing diagram shows
only relative relationships

Fig.A.1 AMline Decoder Timing Diagram
(Latching AMline Data)

A.2.2 Encoder Board - VMEbus

Electrical Specification:

Internal Clock Rate: 1/2 bus clock, 6-8 MHz nominal

Voltage Requirements: +5V

Measured Current Rating: 2.5A at 5V

NOTE: The clock rate must be six times the maximum quadrature encoder signal frequency to ensure reliable counter pulse generation.

Timing Specification: (12MHz System Clock)

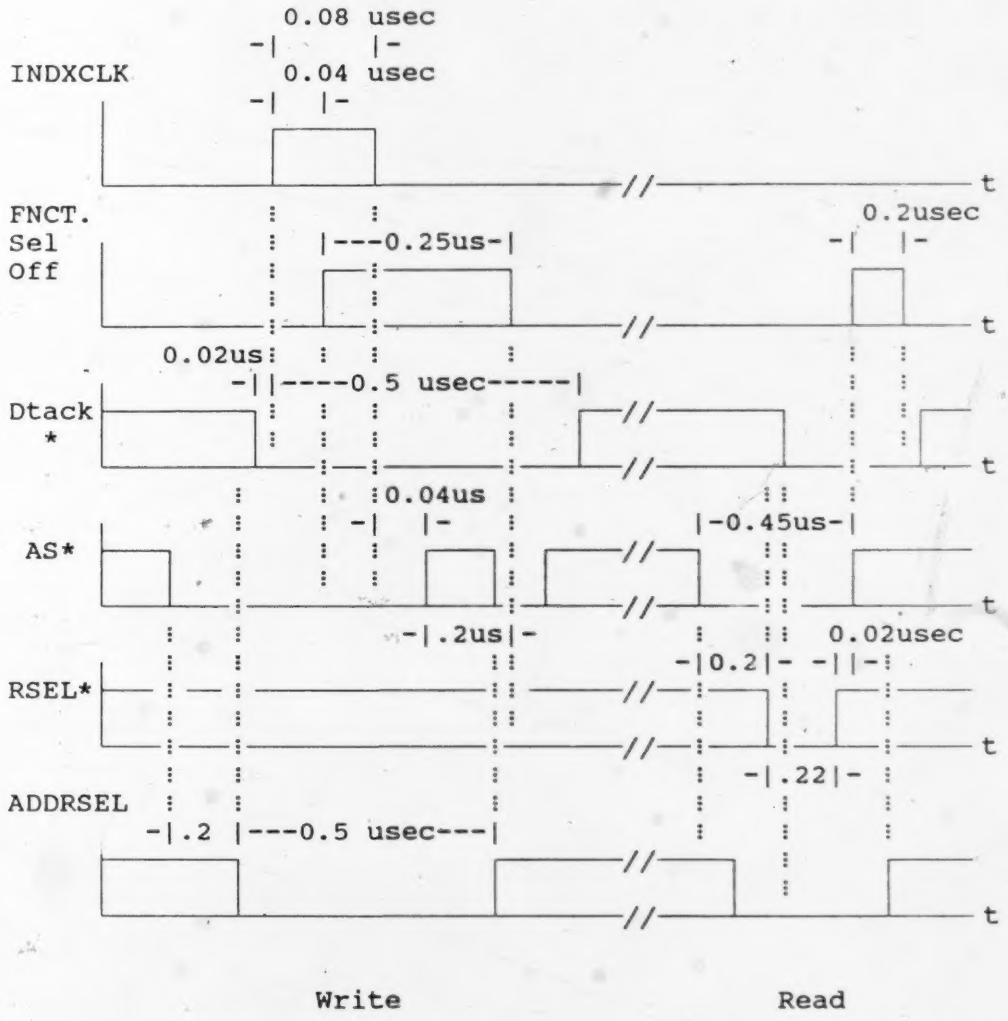

0.08 usec
INDXCLK
0.04 usec
t
FNCT.
Sel
Off
|---0.25us-|
0.2usec
t
0.02us
-||----0.5 usec-----|
Dtack
*
t
0.04us
|-0.45us-|
AS*
t
-|.2us|-
-|0.2|-
0.02usec
RSEL*
t
-|.22|-
ADDRSEL
-|.2
|---0.5 usec---|
t
Write
Read

Fig.A.2 VEB Timing Diagram

(Initializing index load register (write)
and reading from an encoder counter)

A.2.3 PUMA Interface Board (PIB) - Unimate servo bus

Electrical Specification:

Voltage Requirements: +5V, +/-12V

Estimated Current Rating: <1A at 5V or 12V

Measured Signal Noise:

Pot noise at A to D: 60mV p.p.(max.), @ 2.5 MHz

Encoder noise: 100mV p.p.(max.)

D to A servo noise: 50mV p.p.

NOTE: With the arm power on, random noise bursts up to .5V were seen on all signals

DIP Switches: Switches in the ON position denote that a particular joint servo connection is made. Switches connected to grounds must be ON if any joint servos are connected from that switch servo system. The PUMA and VMEbus controller servo lines are connected to different DIP switches. (Refer to Schematics)

A.2.4 Current Amplifier/Sensor Interface

Measured Signal Noise:

Up to 400mV p.p. high frequency

Measured D.C. Offset at Amplifier Output:

approx. -130 mV (signal lower than actual)

A.3 Specifications for U of A / Commercial VMEbus Boards

A.3.1 TUTOR 68000 CPU/RIOT Boards (U of A Developed System)

CPU: 68000

Clock Speed: 12 MHz

Combined Measured Current Rating: 1.5A at 5V

DTACK: Occurs after one clock cycle

SYSCLK: Connected to bus

Address Modifier Lines Supported: AM0..2

RAM Address Space: $000000-00FFFF (64KBytes)

ROM Address Space: $FF0000-$FFFFF

Duart Addresss Space: $EFFC00-$EFFFFF - host terminal
$EFF800-$EFFBFF - remote host

Interrupts: Duart Timer - IRQ 7
Serial Interrupt - IRQ4

A.3.2 DATEL DVME601B A to D Board

CPU: 68010

Clock Speed: 10 MHz

Maximum Measured Current Consumption: 3.1A at 5V

System Address: $FA0000-$FAFFFF (64KBytes)

Local Memory: $200000-$20FFFF (64KBytes)

Address Modifier Codes: $39, $3A, $3D, $3E

System Interrupt: Write Triggered Interrupt - IRQ6

External RS-232: Wired to main terminal, EPROM monitor used

A/D Inputs: Single-ended input with
instrumentational amplifier
set to gain one.

A/D Full Scale Input Range: -10V to +10V all channels

A/D Coding Used: Bipolar, two's complement

Channel Assignment: A/D[0..5] - Robot potentiometers
A/D[6..8] - Joint servo current
A/D[9] - SSRON HI signal
A/D[10] - STOPL signal
A/D[11] - BRK RLS MES

A.3.3 DATEL DVME628V D to A Board

Maximum Measured Current Consumption: 2.3A at 5V

System Address: $FFC900-FFC9FF

DTACK*: 250 nsec delay

Address Modifier Codes: $29, $2D

D/A Full Scale Output Range: +10 to -10V all channels

D/A Coding Used: Bipolar, 2's complement

Channel Assignment: D/A[0..5] - Motor servo signal
D/A[6] - Arm power control,
OFF L signal

A.4 Wiring Specifications

A.4.1 VME/Unimation Controller Cable Wiring Lists

A.4.1.1 Encoder Wiring VMEbus Chassis

Wire #	Description	Color of Pair	DB25 Pin
1	GND	W	15
2	CHA1	R	1
3	CHB1	B	2
4	GND	W	15
5	CHA2	B	3
6	GND	O	15
7	CHB2	GR	4
8	GND	BR	17
9	VCC	R	5
10	CHA3	B	6
11	CHB3	BR	7
12	GND	B	17
13	CHA4	Y	8
14	GND	B	17
15	CHB4	O	9
16	GND	GR	19
17	VCC	GR	10
18	CHA5	W	11
19	CHB5	BL	12
20	GND	B	15
21	CHA6	GR	13
22	GND	B	21
23	CHB6	Y	14
24	GND	GR	21
25	GND	O	15
26	INDX1	R	16
27	GND	BR	17
28	INDX2	R	18
29	GND	Y	19
30	INDX3	R	20
31	GND	BL	21
32	INDX4	R	22
33	GND	BL	23
34	INDX5	GR	24
35	GND	GR	23
36	INDX6	R	25
SHEATH	SHEATH GND	-	23

Table A.3 Encoder Wiring VMEbus Chassis

A.4.1.2 Encoder Wiring Unimate Chassis

Wire #	Description	Color of Pair	DB25 Pin
1	GND	W	21
2	CHA1	R	1
3	CHB1	B	2
4	GND	W	19
5	CHA2	B	3
6	GND	O	15
7	CHB2	GR	4
8	GND	BR	15
9	VCC	R	5
10	CHA3	B	6
11	CHB3	BR	7
12	GND	B	17
13	CHA4	Y	8
14	GND	B	19
15	CHB4	O	9
16	GND	GR	19
17	VCC	GR	10
18	CHA5	W	11
19	CHB5	BL	12
20	GND	B	15
21	CHA6	GR	13
22	GND	B	21/23
23	CHB6	Y	14
24	GND	GR	23/21
25	GND	O	15
26	INDX1	R	16
27	GND	BR	17
28	INDX2	R	18
29	GND	Y	19
30	INDX3	R	20
31	GND	BL	21
32	INDX4	R	22
33	GND	BL	23
34	INDX5	GR	24
35	GND	GR	23
36	INDX6	R	25
SHEATH	SHEATH GND	-	17

Table A.4 Encoder Wiring Unimate Chassis

A.4.1.3 A to D Wiring VMEbus and Unimate Chassis

Wire #	Description	Color of Pair	DB25 Pin
1	CH7	GR	1
2	SIGNAL GND	W	2
3	CH14	W	3
4	SIGNAL GND	B	2
5	CH5	R	4
6	SIGNAL GND	O	5
7	CH12	O	6
8	SIGNAL GND	B	5
9	CH3	BR	7
10	SIGNAL GND	B	8
11	CH10	R	9
12	SIGNAL GND	BR	8
13	CH1	R	10
14	SIGNAL GND	Y	11
15	CH8	R	12
16	SIGNAL GND	BL	13
17	CH15	R	14
18	SIGNAL GND	GR	13
19	CH6	R	15
20	SIGNAL GND	W	16
21	CH13	Y	17
22	SIGNAL GND	B	16
23	CH4	BL	18
24	SIGNAL GND	W	19
25	CH11	O	20
26	SIGNAL GND	GR	19
27	CH2	BL	21
28	SIGNAL GND	B	22
29	CH9	GR	23
30	SIGNAL GND	BR	22
31	CH0	Y	24
32	SIGNAL GND	GR	25
SHEATH	SHEATH GND	-	11

Table A.5 A to D Wiring VMEbus/Unimate Chassis

A.4.1.4 D to A Wiring VMEbus and Unimate Chassis

J1 Pin #	Description	PIB Connector Color of Cable Pair	Pin # - PIB Connection
1	DAC0(+)	R	1
14	DAC0(-)	W	2
4	DAC1(+)	W	3
17	DAC1(-)	B	4
7	DAC2(+)	O	5
20	DAC2(-)	B	6
10	DAC3(+)	R	7
23	DAC3(-)	B	8
24	Sheath GND	-	8

J2 Pin # & Wire	Description	PIB Connector Color of Cable Pair	Pin # - PIB Connection
1 - R	DAC4(+)	BR	9
14 - B	DAC4(-)	B	10
4 - GR	DAC5(+)	Y	11
17 - W	DAC5(-)	B	12
7 - R	DAC6(+)	GR	13
20 - B	DAC6(-)	W	14
10 - GR	DAC7(+)	BL	15
23 - W	DAC7(-)	B	16

NOTE: J1 and J2 connectors are found on the DVME628V D to A board located in the VMEbus chassis.

Table A.6 D to A Wiring VMEbus/Unimate Chassis

A.4.1.5 Current Measurement Wiring PIB

Pin #	Description
1	Motor1(+) - Red
2	Sheath
3	Motor1(-) - Green
4	
5	Motor2(+) - Red
6	Sheath
7	Motor2(-) - Green
8	
9	Motor3(+) - Red
10	Sheath
11	Motor3(-) - Green
12	
13	
14	
15	VMEbus Analog Ground - Black
16	
17	VMEbus Analog Ground - Black
18	
19	VMEbus Analog Ground - Black
20	
21	VMEbus Analog Ground - Black
22	
23	VMEbus Analog Ground - Black
24	
25	VMEbus Analog Ground - Black

Table A.7 Current Measurement Wiring PIB (DB25)

A.4.1.6 Current Measurement Sensor Board Wiring

Pin #	Description
1	Motor1(+) - Red
2	VAG+VAG+Sheath - Black + White
3	Motor2(+) - Red
4	VAG+VAG+Sheath - Black + White
5	Motor3(+) - Red
6	Motor1(-) - Green
7	VAG+VAG+Sheath - Black + White
8	Motor2(-) - Green
9	Motor3(-) - Green

Table A.8 Current Measurement Sensor Board Wiring

A.4.2 PUMA Slot #50 Wiring List

TO: (SLOT #50, PIB)	FROM:
	SLOT #56 - Power Amp. Control - Sch.305-0980
AA1 - SSRON HI	AU1 - 12.5V Output Pin
AB1 - STOP L	AR2 - TTL
AC1 - BRK RLS MES	AN2 - TTL
AD1 - OFF L	AV1 - TTL
	Slot #58 - Cable Card - Sch.305-1031
AE1 - ENCA 1	AA1 - 1 ENCA
AF1 - POT 1	AF1 - 1 POT
AH1 - ENCI 2	AM1 - 2 ENCI
AJ1 - GND	
AK1 - ENCA 2	AH1 - 2 ENCA
AL1 - ENCA 3	AP1 - 3 ENCA
AM1 - GND	
AN1 - ENCI 4	BE1 - 4 ENCI
AP1 - ENCA 4	BA1 - 4 ENCA
AR1 - ENCA 5	BH1 - 5 ENCA
AS1 - Not Connected	
AT1 - GND	
AU1 - ENCA 6	BP1 - 6 ENCA
AV1 - Not Connected	
BA1 - ENCQ 6	BS1 - 6 ENCQ
AA2 - +5V	
AB2 - -12V	
AC2 - GND	
AD2 - +12V	
AE2 - ENCQ 1	AC1 - 1 ENCQ
AF2 - ENCI 1	AE1 - 1 ENCI
AH2 - POT 2	AN1 - 2 POT
AJ2 - ENCQ 2	AK1 - 2 ENCQ
AK2 - POT 3	AV1 - POT 3
AL2 - ENCI 3	AU1 - 3 ENCI
AM2 - ENCQ 3	AS1 - 3 ENCQ
AN2 - POT 4	BF1 - 4 POT
AP2 - ENCQ 4	BC1 - 4 ENCQ
AR2 - ENCQ 5	BK1 - 5 ENCQ
AS2 - POT 5	BN1 - 5 POT
AT2 - ENCI 5	BM1 - 5 ENCI
AU2 - POT 6	BV1 - 6 POT
AV2 - ENCI 6	BU1 - 6 ENCI

TO: (SLOT #50, PIB)	FROM:
BA1 - SEE ABOVE	
BB1 - -12V	
	SLOT #56
	- Power Amp. Control
	- Sch.305-0980
"OUT" to motor servos	
BC1 - DAC(-) 1	AE2 - DAC- JT1, 4 J73
BD1 - DAC(+) 1	AE1 - DAC+ JT1, 3 J73
BE1 - DAC(-) 2	AF2 - DAC- JT2, 2 J73
BF1 - DAC(+) 2	AF1 - DAC+ JT2, 1 J73
BH1 - DAC(-) 3	AH2 - DAC- JT3, 8 J73
BJ1 - GND	
BK1 - DAC(+) 3	AH1 - DAC+ JT3, 5 J73
BA2 - +5V	
BB2 - -12V	
BC2 - GND	
BD2 - +12V	
BE2 - DAC(-) 4	AJ2 - DAC- JT4, 4 J74
BF2 - DAC(+) 4	AJ1 - DAC+ JT4, 3 J74
BH2 - DAC(-) 5	AK2 - DAC- JT5, 2 J74
BJ2 - DAC(+) 5	AK1 - DAC+ JT5, 1 J74
BK2 - DAC(-) 6	AL2 - DAC- JT6, 8 J74
BL2 - DAC(+) 6	AL1 - DAC+ JT6, 5 J74
"IN" - from 6503 processors	
BL1 - DAC(-) 1	SLOT#40, AC1 - DAC-,JT1
BM1 - GND	
BN1 - DAC(+) 1	SLOT#40, AS2 - DAC+,JT1
BP1 - DAC(-) 2	SLOT#41, AC1 - DAC-,JT2
BR1 - DAC(+) 2	SLOT#41, AS2 - DAC+,JT2
BS1 - Not Connected	
BT1 - GND	
BU1 - DAC(+) 3	SLOT#42, AS2 - DAC+,JT3
BV1 - +5V	
BM2 - DAC(-) 4	SLOT#43, AC1 - DAC-,JT4
BN2 - DAC(+) 4	SLOT#43, AS2 - DAC+,JT4
BP2 - DAC(-) 5	SLOT#44, AC1 - DAC-,JT5
BR2 - DAC(+) 5	SLOT#44, AS2 - DAC+,JT5
BS2 - DAC(-) 6	SLOT#45, AC1 - DAC-,JT6
BT2 - DAC(+) 6	SLOT#45, AS2 - DAC+,JT6
BU2 - +12V	
BV2 - DAC(-) 3	SLOT#42, AC1 - DAC-,JT3

Table A.9 PUMA Slot #50 Wiring List

A.5 Miscellaneous Measurements

A.5.1 PUMA Pot Voltage To Index Pulse Resolution

Joint Number	Voltage Between Index Pulses
1	43.5mV
2	41.1mV
3	38.4mV
4	39.1mV
5	43.1mV
6	42.8mV

Table A.10 Pot/Index Pulse Resolution

A.5.2 PUMA System Status Signals

SS RON HI:

When this signal is low (less than 0.2 volts) a temperature fault, a bad 40 V power supply condition, a brake released manually condition, an LSI-11 power off signal or an OFF-L signal has been detected. Such conditions are considered to be faults and therefore turn the arm power off.

STOP L:

When this signal is low the arm power has been turned off.

BRK RLS MES:

When this signal is high the robot arm safety brakes have been released manually and the robot arm will fall unless arm power is on or it has been mechanically supported.

A.5.3 D.C. Resistance of PUMA 560 Joint Motors

The positive and negative leads of each of the PUMA 560 D.C. motors were individually connected to a multimeter and the following values were obtained:

Joint Number	Motor Resistance
1	2 ohms
2	3 ohms
3	2.9 ohms
4	11 ohms
5	7.2 ohms
6	49.4 ohms

Table A.11 Joint Motor Resistance Values

A.6 System Schematics

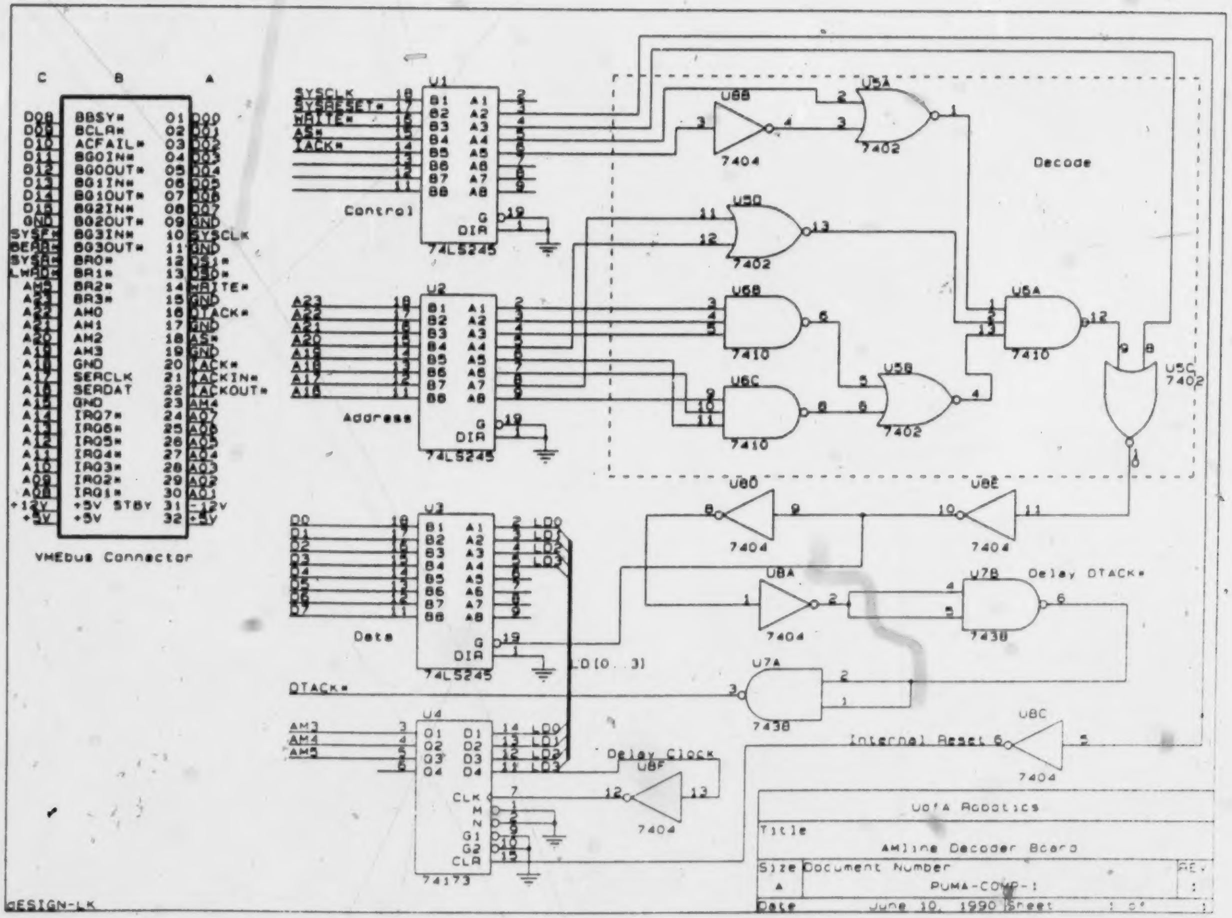

C
B
A
BBSY*
BCLR*
ACFAIL*
BG0IN*
BG0OUT*
BG1IN*
BG1OUT*
BG2IN*
BG2OUT*
BG3IN*
BG3OUT*
BR0*
BR1*
BR2*
BR3*
AM0
AM1
AM2
AM3
GND
SERCLK
SERDAT
GND
IRQ7*
IRQ6*
IRQ5*
IRQ4*
IRQ3*
IRQ2*
IRQ1*
+5V STBY
+5V
SYSCLK
DS1*
DS0*
WRITE*
DTACK*
AS*
IACK*
IACKIN*
IACKOUT*
AM4
+12V
-12V
+5V
VMEbus Connector
SYSCLK
SYSRESET*
WRITE*
AS*
IACK*
Control
U1
74LS245
A23
A22
A21
A20
A19
A18
A17
A16
Address
U2
74LS245
D0
D1
D2
D3
D4
D5
D6
D7
Data
U3
74LS245
LD0
LD1
LD2
LD3
LD(0..3)
DTACK*
AM3
AM4
AM5
U4
74173
CLK
M
N
G1
G2
CLR
G
DIR
Decode
U8B
7404
U5A
7402
U5D
7402
U6B
7410
U6C
7410
U5B
7402
U5A
7410
U5C
7402
U8D
7404
U8E
7404
U8A
7404
U7B
7438
Delay DTACK*
U7A
7438
Delay Clock
U8F
7404
Internal Reset
U8C
7404
UofA Robotics
Title
AMline Decoder Board
Size
A
Document Number
PUMA-COMP-1
REV
Date
June 10, 1990
Sheet
DESIGN-LK

Figure A.3 AMline Decoder Board

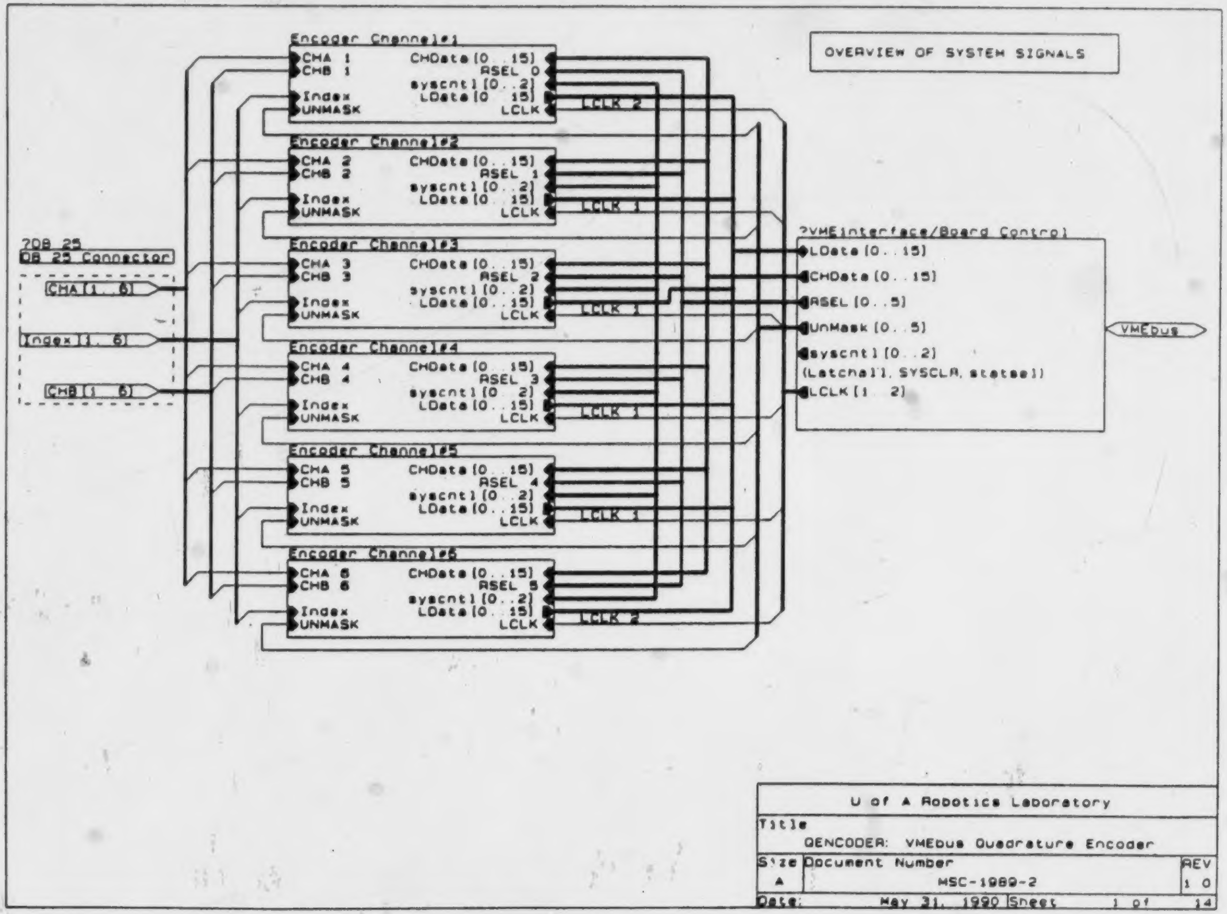

OVERVIEW OF SYSTEM SIGNALS
?DB 25
DB 25 Connector
CHA[1..6]
Index[1..6]
CHB[1..6]
Encoder Channel#1
CHA 1
CHB 1
Index
UNMASK
CHData[0..15]
RSEL 0
syscntl[0..2]
LData[0..15]
LCLK
LCLK 2
Encoder Channel#2
CHA 2
CHB 2
RSEL 1
LCLK 1
Encoder Channel#3
CHA 3
CHB 3
RSEL 2
Encoder Channel#4
CHA 4
CHB 4
RSEL 3
Encoder Channel#5
CHA 5
CHB 5
RSEL 4
Encoder Channel#6
CHA 6
CHB 6
RSEL 5
?VMEinterface/Board Control
LData[0..15]
CHData[0..15]
RSEL[0..5]
UnMask[0..5]
syscntl[0..2]
(Latchall, SYSCLR, statsel)
LCLK[1..2]
VMEbus
U of A Robotics Laboratory
Title
QENCODER: VMEbus Quadrature Encoder
Size
A
Document Number
MSC-1989-2
REV
1 0
Date:
May 31, 1990
Sheet
1 of 14

Figure A.4 Encoder Board: Block Diagram

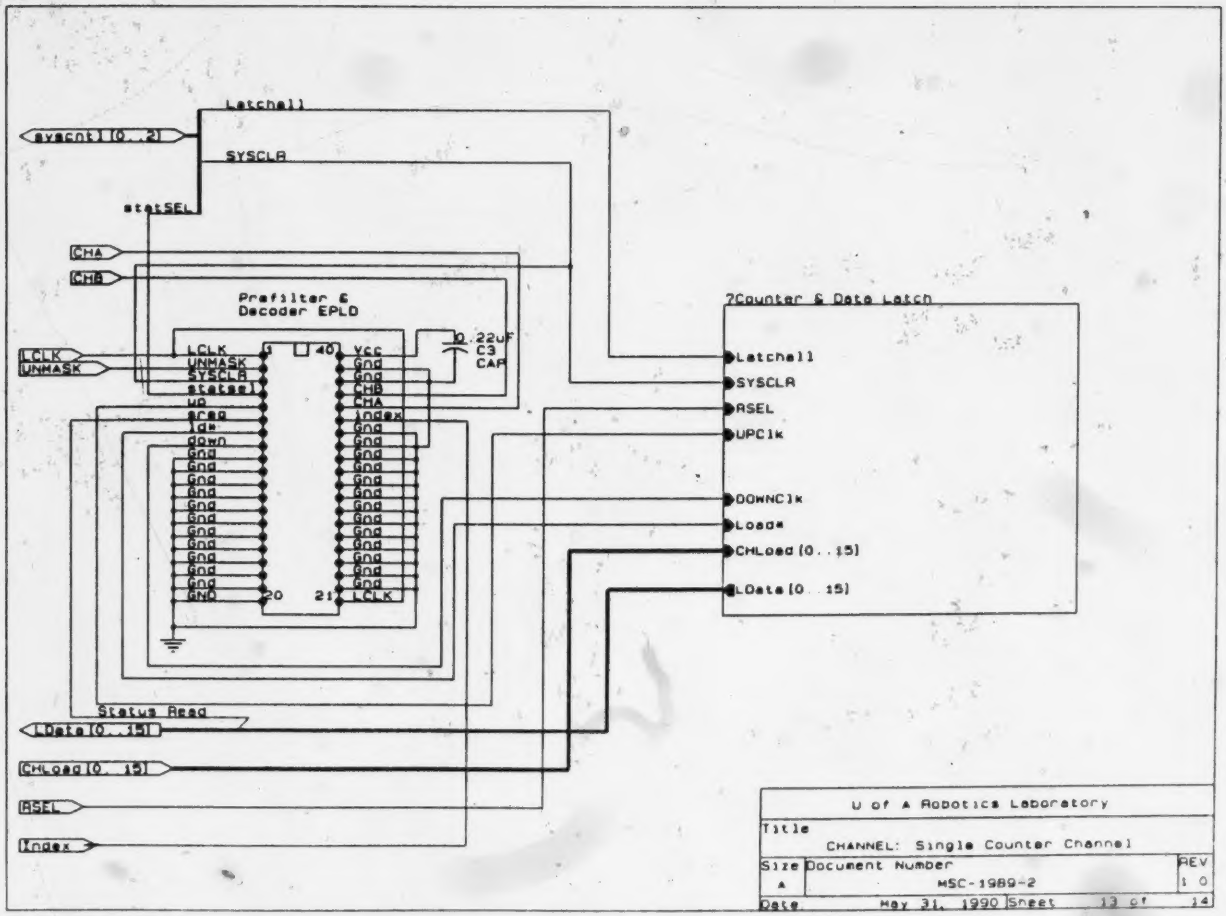
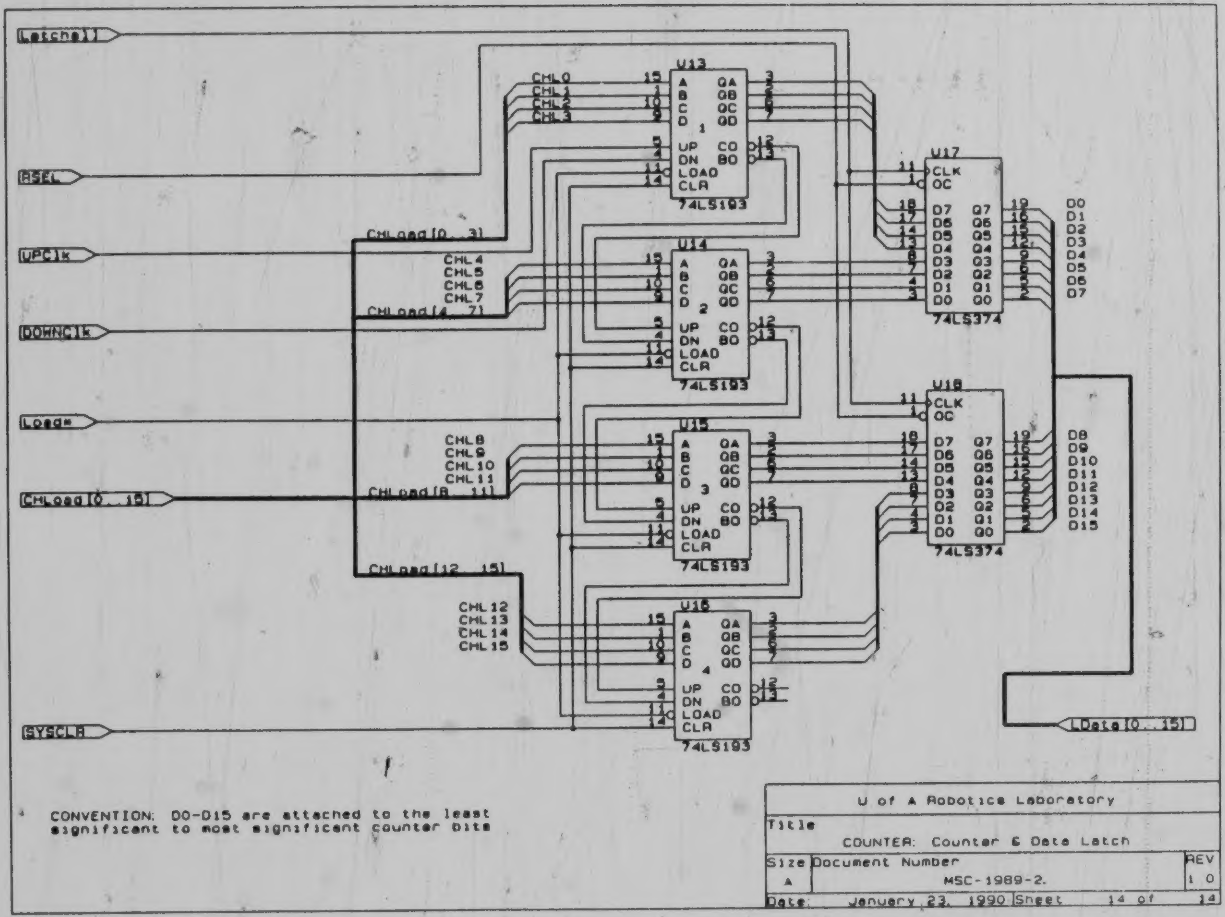

sysant1 [0..2]
Latchall
SYSCLR
statSEL
CHA
CHB
Prefilter &
Decoder EPLD
LCLK
UNMASK
SYSCLR
statsel
up
areg
ld*
down
Gnd
GND
1
20
40
21
Vcc
CHB
CHA
index
LCLK
0.22uF
C3
CAP
Status Read
LData [0..15]
CHLoad [0..15]
RSEL
Index
?Counter & Data Latch
Latchall
SYSCLR
RSEL
UPClk
DOWNClk
Load*
CHLoad [0..15]
LData [0..15]
U of A Robotics Laboratory
Title
CHANNEL: Single Counter Channel
Size
A
Document Number
MSC-1989-2
REV
1.0
Date
May 31, 1990
Sheet 13 of 14

Figure A.5 Encoder Board: Quadrature Channel


Latchall
RSEL
UPClk
DOWNClk
Load*
CHLoad[0..15]
SYSCLR
CHLoad[0..3]
CHLoad[4..7]
CHLoad[8..11]
CHLoad[12..15]
CHL0
CHL1
CHL2
CHL3
CHL4
CHL5
CHL6
CHL7
CHL8
CHL9
CHL10
CHL11
CHL12
CHL13
CHL14
CHL15
U13
U14
U15
U16
74LS193
A
B
C
D
UP
DN
LOAD
CLR
QA
QB
QC
QD
CO
BO
1
2
3
4
U17
U18
CLK
OC
D7
D6
D5
D4
D3
D2
D1
D0
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
74LS374
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
LData[0..15]

CONVENTION: D0-D15 are attached to the least significant to most significant counter bits

U of A Robotics Laboratory

Title

COUNTER: Counter & Data Latch

Size A

Document Number MSC-1989-2.

REV 1 0

Date: January 23, 1990

Sheet 14 of 14

Figure A.6 Encoder Board: Channel Counter & Latch

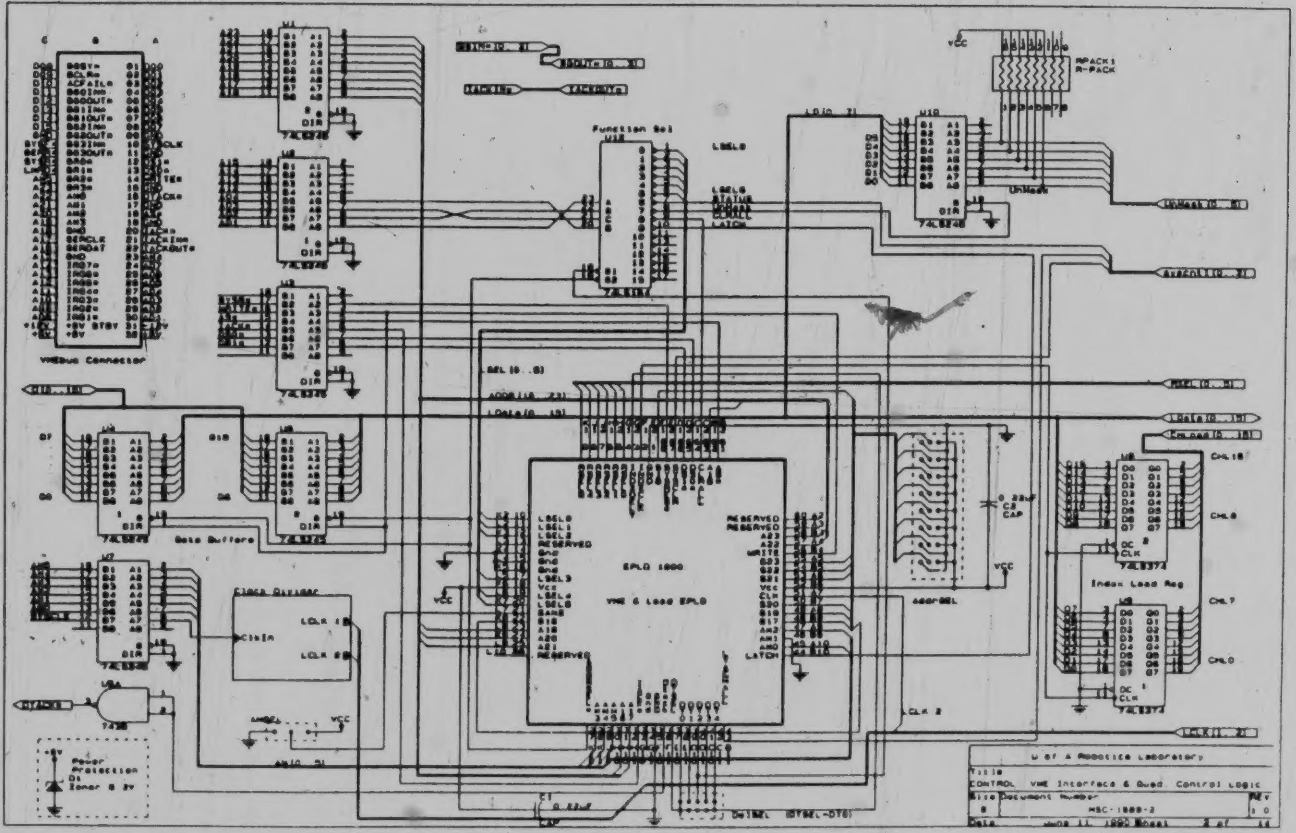

VMEbus Connector
Function Sel
Data Buffers
Clock Divider
LCLK 1
LCLK 2
EPLD 1800
VME & Load EPLD
LSEL0
LSEL1
LSEL2
RESERVED
VCC
LATCH
WRITE
AddrSEL
Index Load Reg
74LS245
74LS374
74LS138
UnMask
RPACK1
R-PACK
CHL15
CHL8
CHL7
CHL0
LCLK 2
Power Protection
Zener 6.2V
C1
C2
0.22uF
CAP
U of A Robotics Laboratory
Title
CONTROL VME Interface & Quad. Control Logic
Size
Document Number
B
MSC-1989-2
REV
1.0
Date
June 11, 1990
Sheet
2 of 14

Figure A.7 Encoder Board: Main Control Logic

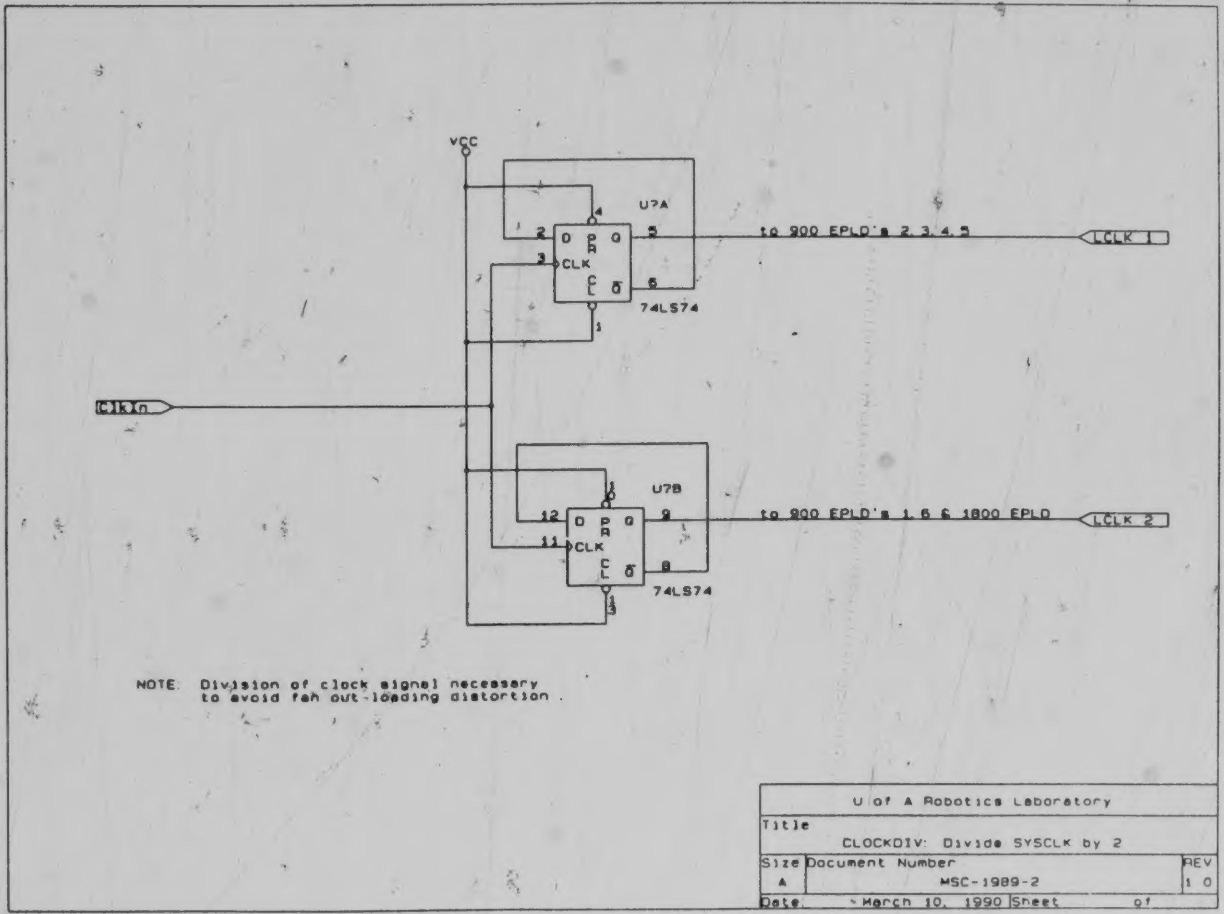
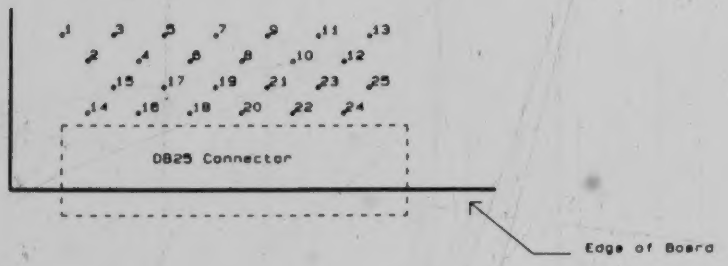

VCC
U7A
74LS74
D
PR
Q
CLK
CL
Q̄
2
3
4
5
6
1
to 900 EPLD's 2,3,4,5
LCLK 1
U7B
74LS74
12
11
10
9
8
13
to 900 EPLD's 1,6 & 1800 EPLD
LCLK 2
ClkIn
NOTE: Division of clock signal necessary
to avoid fan out-loading distortion.
U of A Robotics Laboratory
Title
CLOCKDIV: Divide SYSCLK by 2
Size
A
Document Number
MSC-1989-2
REV
1.0
Date:
March 10, 1990
Sheet
of

Figure A.8 Encoder Board: Clock Divider

DB25 BOARD CONNECTOR
(Bottom View)


1
3
5
7
9
11
13
2
4
6
8
10
12
15
17
19
21
23
25
14
16
18
20
22
24
DB25 Connector
Edge of Board

Pin#	Description
1	CHA 1
2	CHB 1
3	CHA 2
4	CHB 2
5	VCC
6	CHA 3
7	CHB 3
8	CHA 4
9	CHB 4
10	VCC
11	CHA 5
12	CHB 5
13	CHA 6
14	CHB 6
15	GND
16	INDX 1
17	GND
18	INDX 2
19	GND
20	INDX 3
21	GND
22	INDX 4
23	GND
24	INDX 5
25	INDX 6

U of A Robotics Laboratory
Title
DB25con: External DB25 Connection
Size A
Document Number MSC-1989-2
REV 1 0
Date: May 31, 1990
Sheet of

Figure A.9 Encoder Board: External Connector

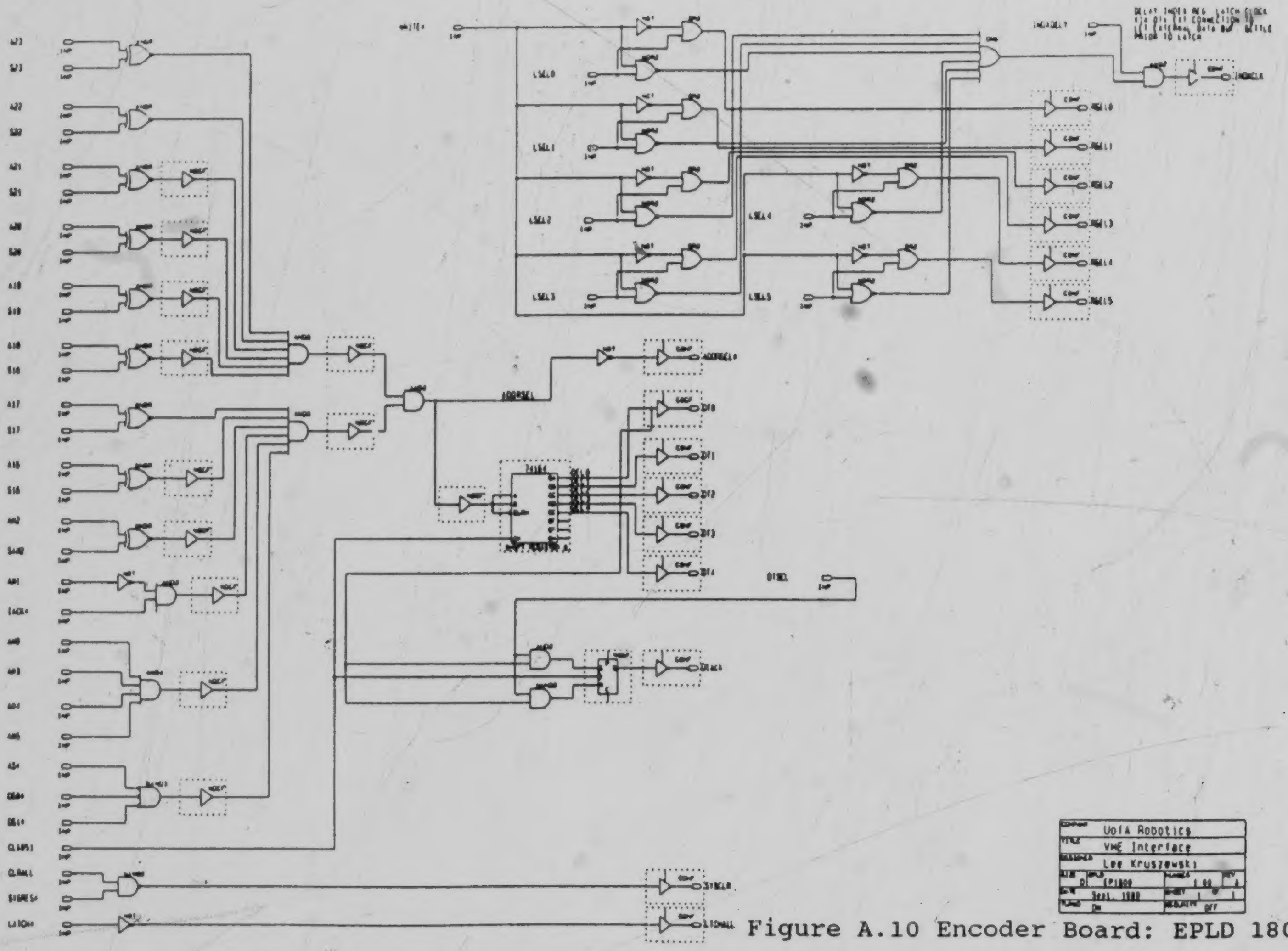

A23
S23
A22
S22
A21
S21
A20
S20
A19
S19
A18
S18
A17
S17
A16
S16
AM2
SAM2
AM1
IACK*
AM0
AM3
AM4
AM5
AS*
DS0*
DS1*
CLRALL
SYSRES*
LATCH*
WRITE*
LSEL0
LSEL1
LSEL2
LSEL3
LSEL4
LSEL5
RSEL0
RSEL1
RSEL2
RSEL3
RSEL4
RSEL5
ADDRSEL
ADDRSEL*
74164
DTSEL
UofA Robotics
VME Interface
Lee Kruszewski
EP1800
1.00
A
1 of 1

Figure A.10 Encoder Board: EPLD 1800 Logic

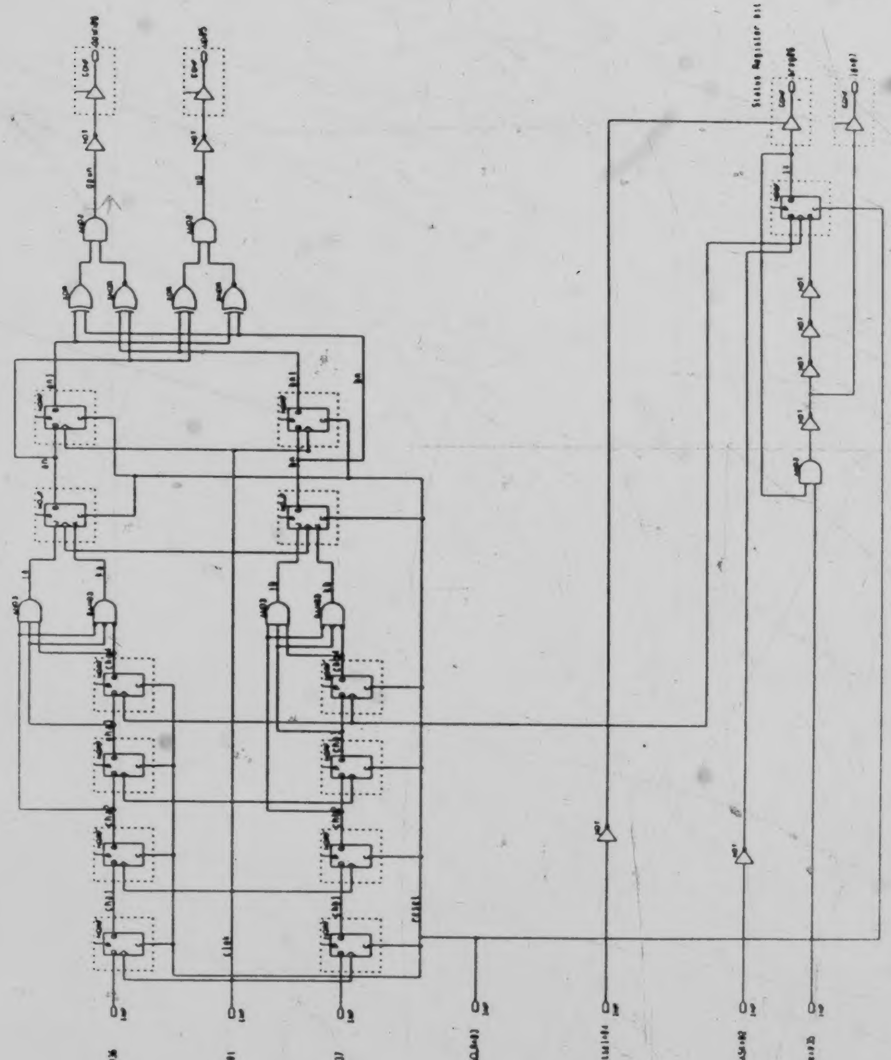

Status Register Bit
U of A Robotics
FILTER
Lee Kruszewski

Figure A.11 Encoder Board: EPLD 900 Logic

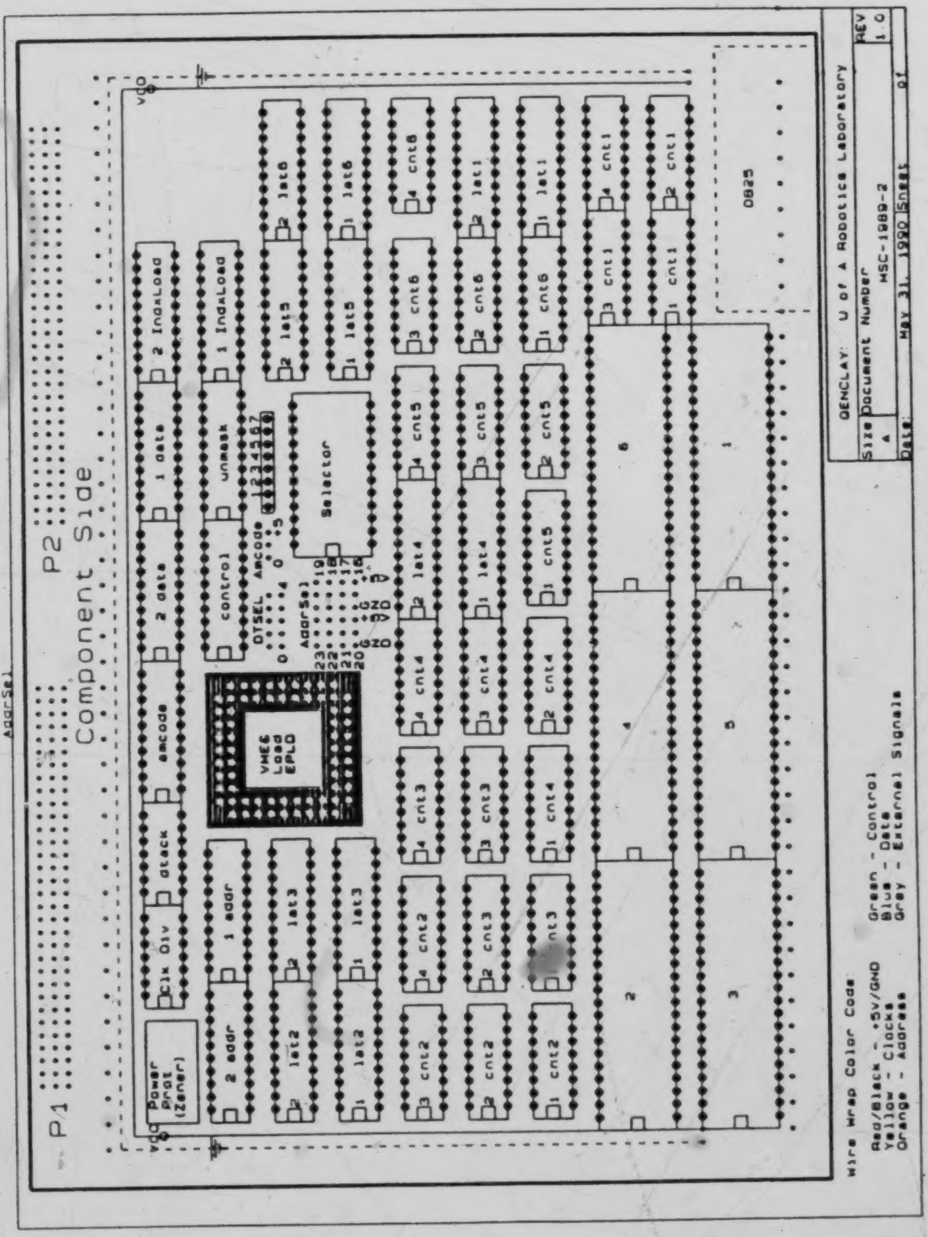

P1
P2
Component Side
AddrSel
VDD
Power Prot (Zener)
Clk Div
dtack
amcode
2 data
1 data
2 IndxLoad
1 IndxLoad
2 addr
1 addr
control
unmask
lat2
lat3
VMEG Load EPLD
DTSEL
AMcode
AddrSel
Selector
cnt2
cnt3
cnt4
cnt5
cnt6
lat4
lat5
lat6
lat1
cnt1
0825
Wire Wrap Color Code
Red/Black - +5V/GND
Yellow - Clocks
Orange - Address
Green - Control
Blue - Data
Grey - External Signals
GENCLAY: U of A Robotics Laboratory
Size
A
Document Number
MSC-1989-2
REV
1.0
Date: May 31, 1990
Sheet of

Figure A.12 Encoder Board Layout

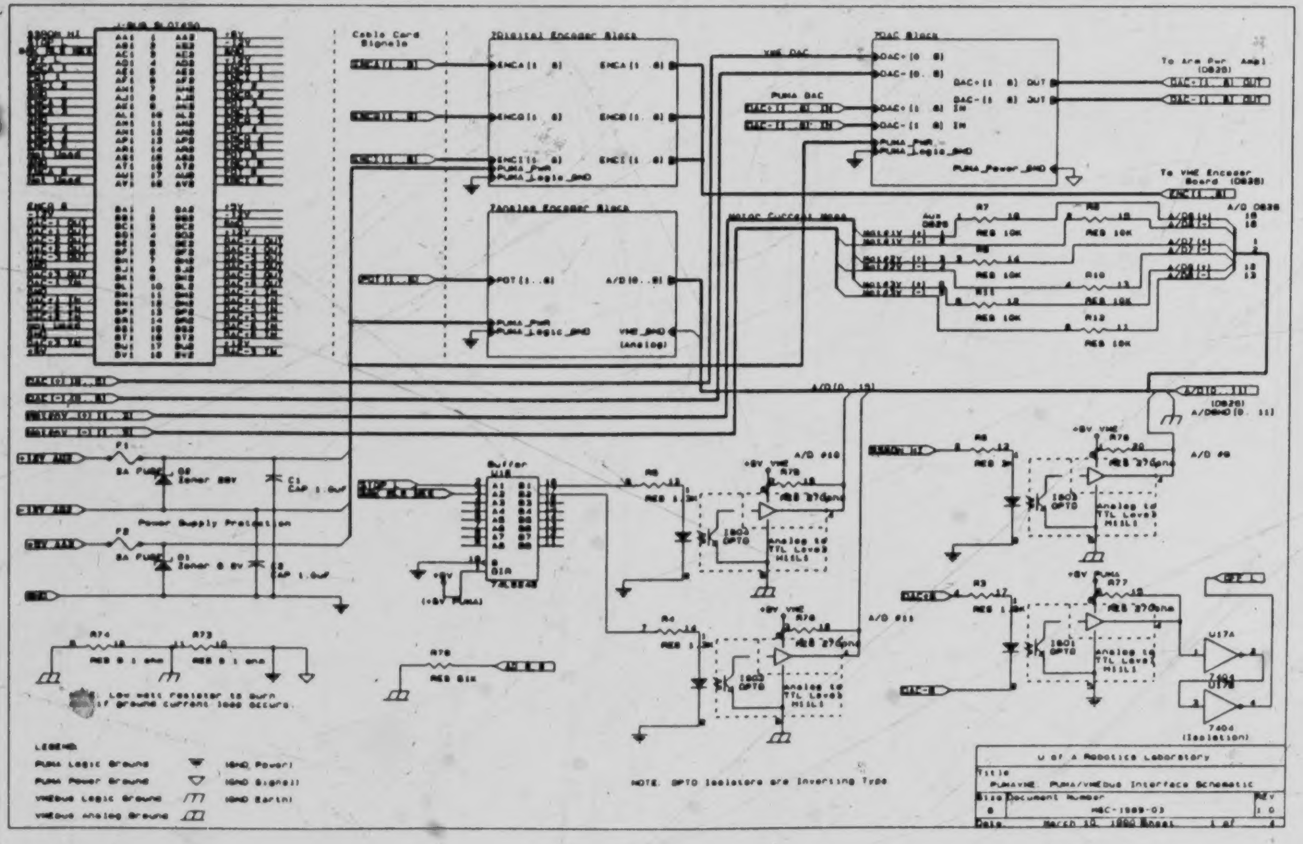

Cable Card Signals
Digital Encoder Block
Analog Encoder Block
DAC Block
VME DAC
PUMA DAC
To Arm Pwr. Ampl (DB25)
To VME Encoder Board (DB25)
Motor Current Mons
A/D DB38
Buffer
U18
74LS245
Power Supply Protection
5A FUSE
Zener 6.8V
CAP 1.0uF
A/D #8
A/D #9
A/D #10
A/D #11
OPTO
Analog to TTL Level
H11L1
7404 (Isolation)
RES 10K
RES 270ohm
R74
R73
RES 0.1 ohm
R78
RES 51K
Low watt resistor to burn if ground current loop occurs.
LEGEND
PUMA Logic Ground (GND Power)
PUMA Power Ground (GND Signal)
VMEbus Logic Ground (GND Earth)
VMEbus Analog Ground
NOTE: OPTO Isolators are Inverting Type
U of A Robotics Laboratory
Title
PUMAVME: PUMA/VMEbus Interface Schematic
Size
B
Document Number
RSC-1989-03
REV
1.0
Date
March 10, 1990
Sheet
1 of 1

Figure A.13 PIB Block Diagram

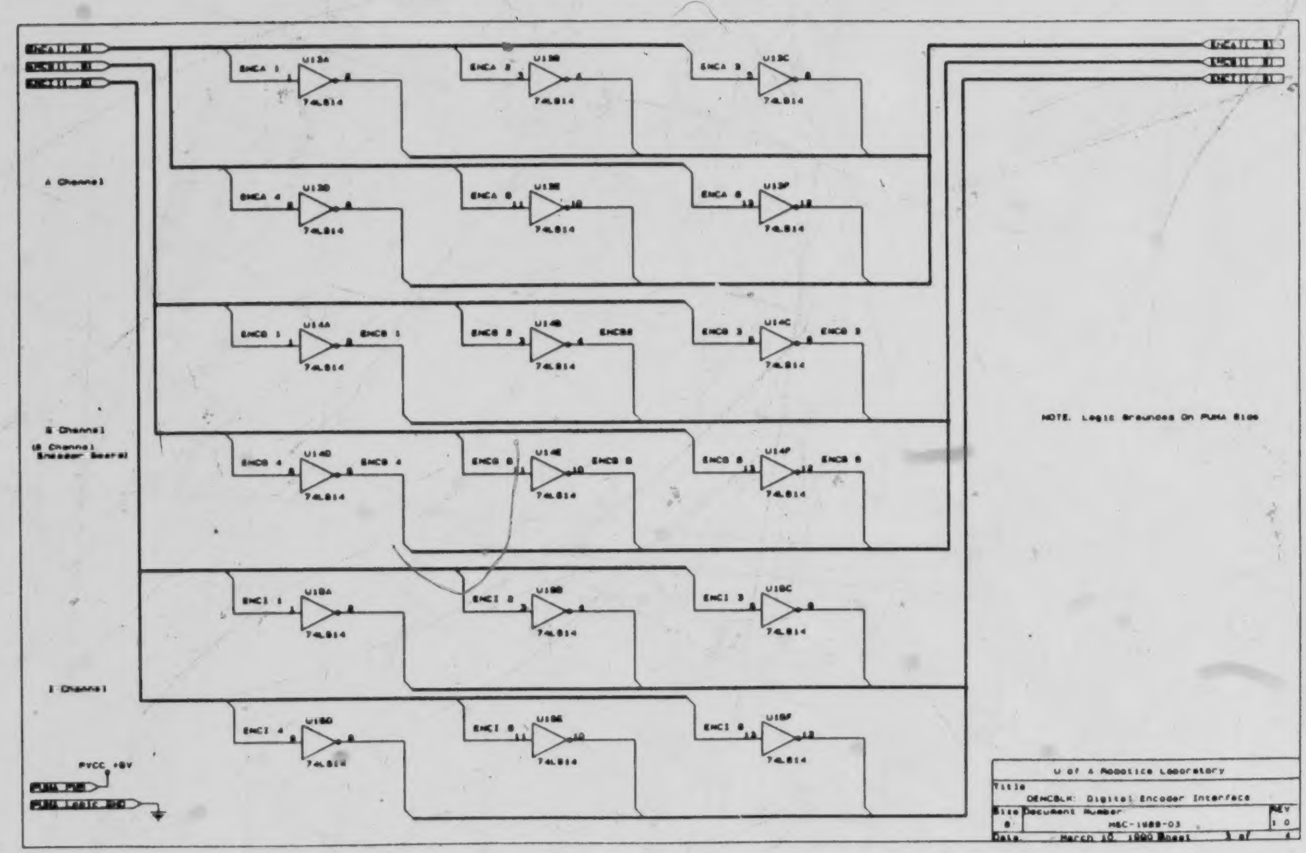

A Channel
B Channel
(B Channel
Encoder Board)
I Channel
PVCC +5V
PUMA PWR
PUMA Logic GND
ENCA 1
ENCA 2
ENCA 3
ENCA 4
ENCA 5
ENCA 6
ENCB 1
ENCB 2
ENCB 3
ENCB 4
ENCB 5
ENCB 6
ENCI 1
ENCI 2
ENCI 3
ENCI 4
ENCI 5
ENCI 6
U13A
U13B
U13C
U13D
U13E
U13F
U14A
U14B
U14C
U14D
U14E
U14F
U15A
U15B
U15C
U15D
U15E
U15F
74LS14
NOTE: Logic Grounded On PUMA Side
U of A Robotics Laboratory
Title
DENCBLK: Digital Encoder Interface
Size
B
Document Number
MSC-1989-03
REV
1.0
Date
March 10, 1990
Sheet
3 of 4

Figure A.14 PIB: Digital Encoder Buffering

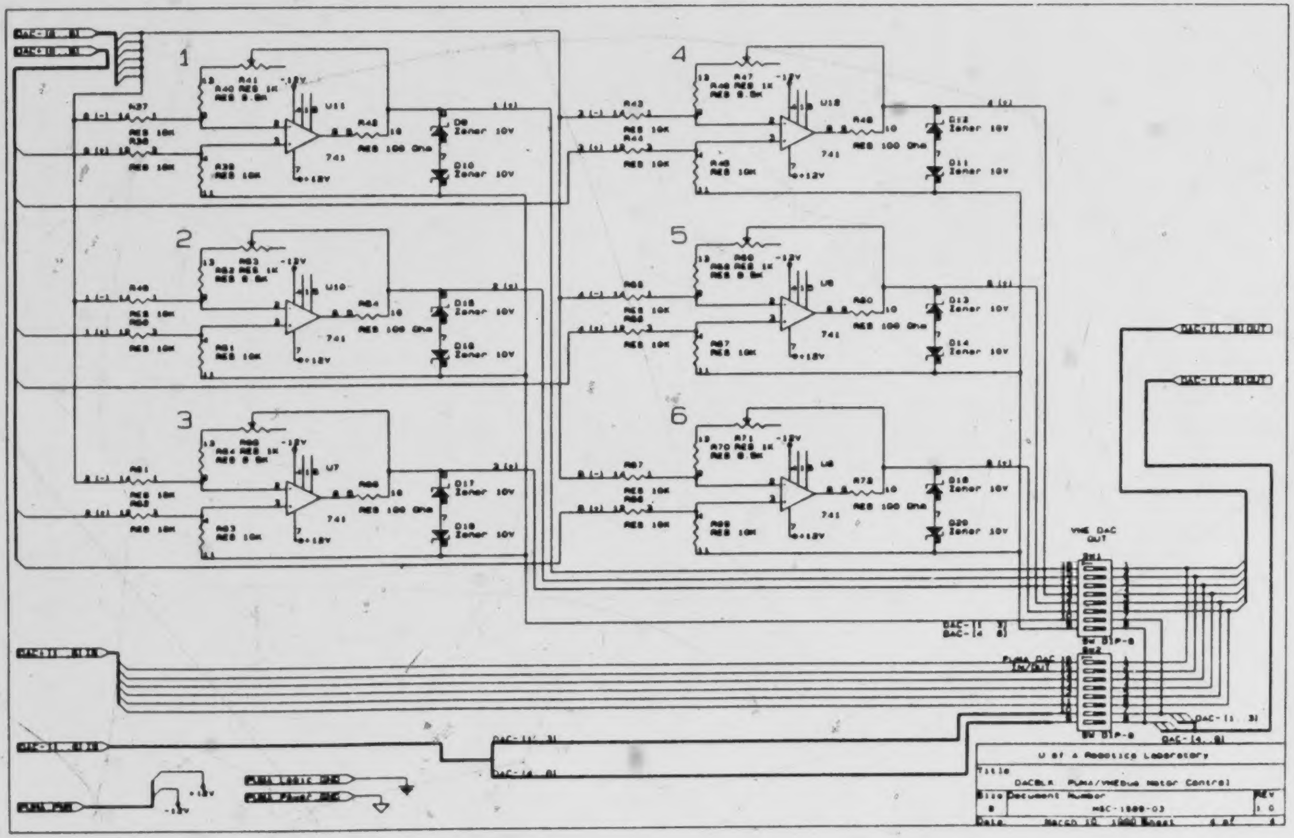

1
2
3
4
5
6
741
Zener 10V
VME DAC OUT
SW DIP-8
PUMA DAC IN/OUT
DAC-(1..3)
DAC-(4..8)
U of A Robotics Laboratory
Title
DACBLK PUMA/VMEbus Motor Control
Size
B
Document Number
MSC-1988-03
REV
1.0
Date
March 10, 1988
Sheet

Figure A.15 PIB: Motor Servo Command Buffering

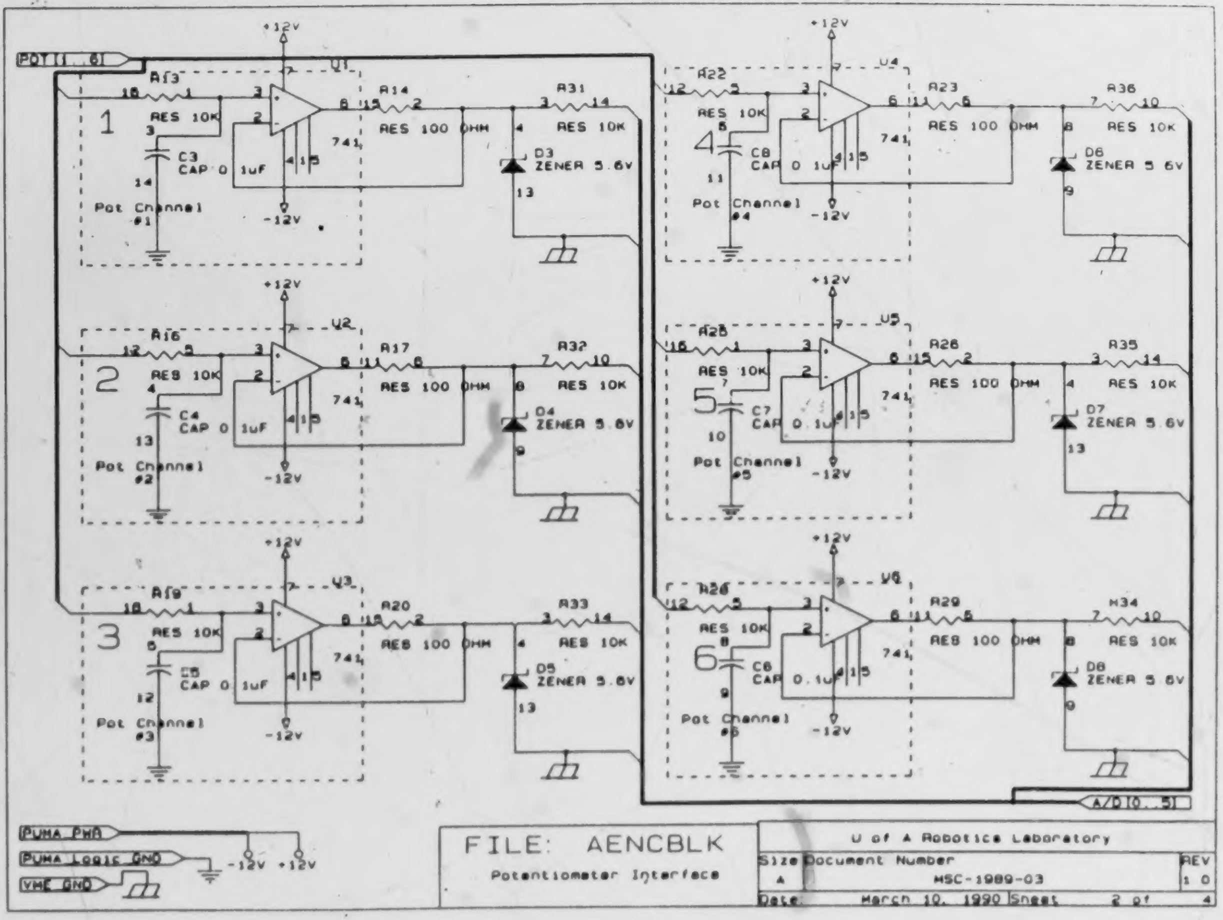

POT[1..6]
A/D[0..5]
PUMA PWR
PUMA Logic GND
VME GND
-12V +12V
U1
U2
U3
U4
U5
U6
741
+12V
-12V
RES 10K
RES 100 OHM
ZENER 5.6V
CAP 0.1uF
C3
C4
C5
C6
C7
C8
D3
D4
D5
D6
D7
D8
R13
R14
R31
R16
R17
R32
R19
R20
R33
R22
R23
R36
R25
R26
R35
R28
R29
R34
1
2
3
4
5
6
Pot Channel #1
Pot Channel #2
Pot Channel #3
Pot Channel #4
Pot Channel #5
Pot Channel #6
FILE: AENCBLK
Potentiometer Interface
U of A Robotics Laboratory
Size
A
Document Number
MSC-1989-03
REV
1.0
Date
March 10, 1990
Sheet
2 of 4

Figure A.16 PIB: Joint Potentiometer Buffering

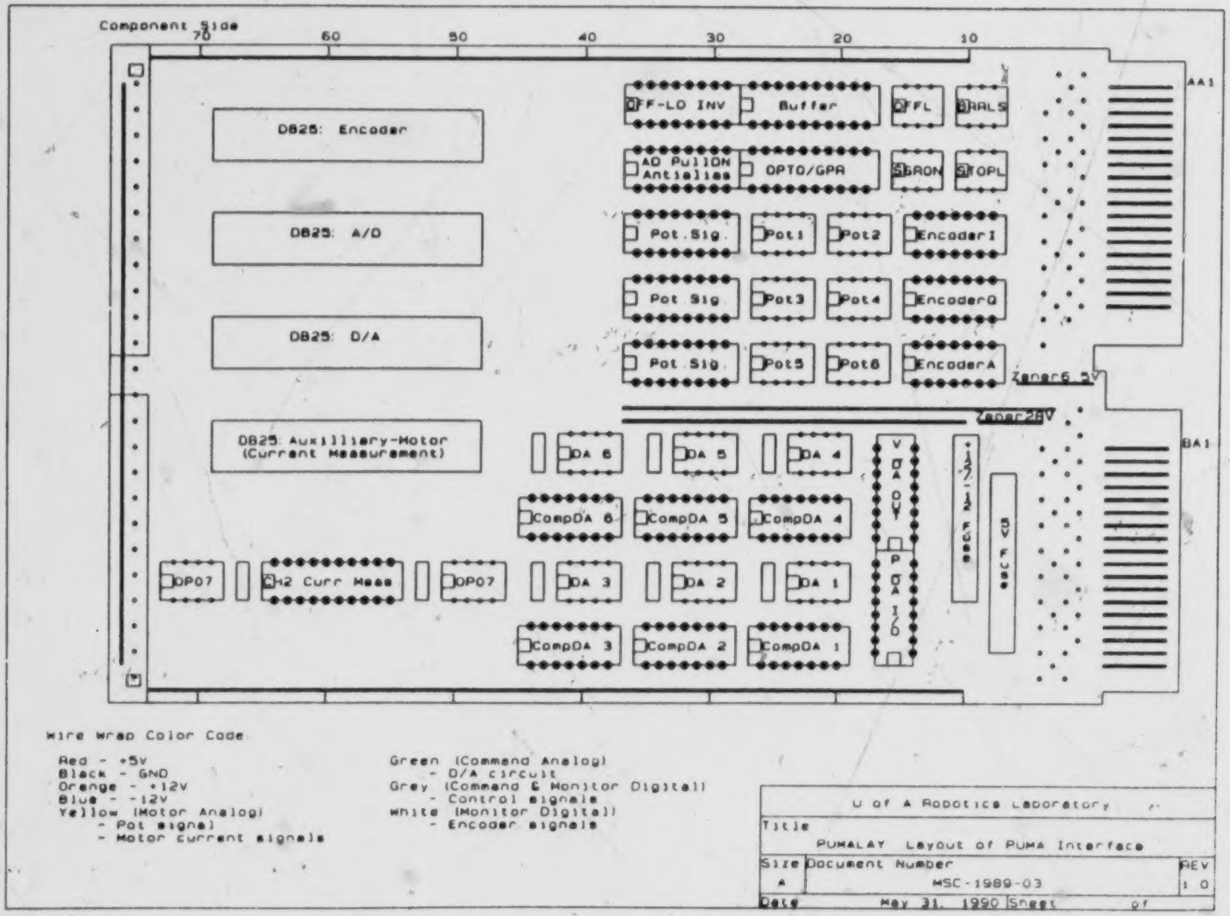

Component Side
70
60
50
40
30
20
10
AA1
BA1
DB25: Encoder
DB25: A/D
DB25: D/A
DB25: Auxilliary-Motor (Current Measurement)
OFF-LO INV
Buffer
OFFL
BRALS
AD PullON Antialias
OPTO/GPA
SBRON
STOPL
Pot.Sig.
Pot1
Pot2
EncoderI
Pot Sig
Pot3
Pot4
EncoderQ
Pot.Sig.
Pot5
Pot6
EncoderA
Zener6.5V
Zener28V
DA 6
DA 5
DA 4
V DA OUT
+12/-12 Fuse
5V Fuse
CompDA 6
CompDA 5
CompDA 4
OP07
CH2 Curr Meas
OP07
DA 3
DA 2
DA 1
P DA I/O
CompDA 3
CompDA 2
CompDA 1
Wire Wrap Color Code
Red - +5V
Black - GND
Orange - +12V
Blue - -12V
Yellow (Motor Analog)
- Pot signal
- Motor current signals
Green (Command Analog)
- D/A circuit
Grey (Command & Monitor Digital)
- Control signals
White (Monitor Digital)
- Encoder signals
U of A Robotics Laboratory
Title
PUMALAY Layout of PUMA Interface
Size
A
Document Number
MSC-1989-03
REV
1.0
Date
May 31, 1990
Sheet
of

Figure A.17 PIB Layout

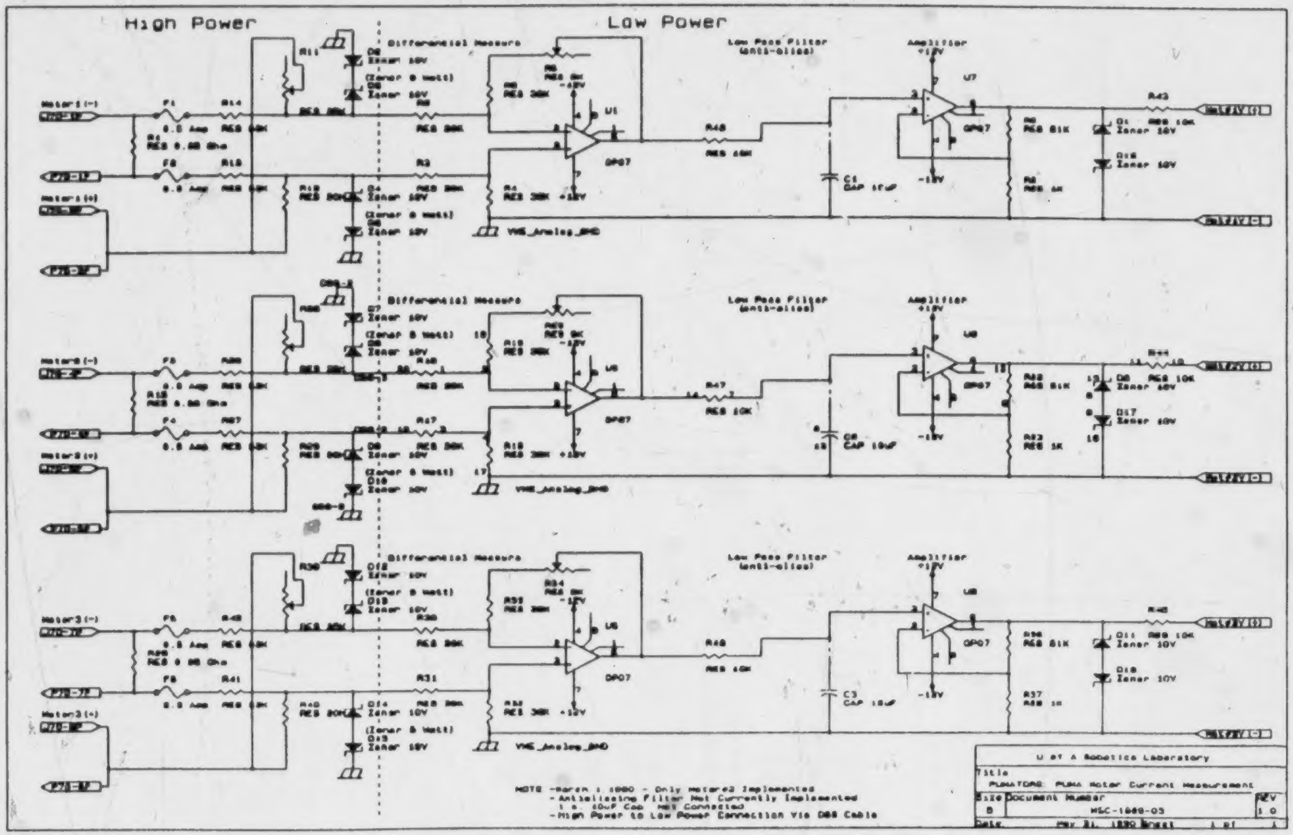

High Power
Low Power
Differential Measure
Low Pass Filter
(anti-alias)
Amplifier
OP07
VME_Analog_GND
Motor1(-)
Motor1(+)
Motor2(-)
Motor2(+)
Motor3(-)
Motor3(+)
0.5 Amp
Zener 10V
(Zener 5 Watt)
CAP 10uF
NOTE -March 1 1990 - Only Motor#2 Implemented
- Antialiasing Filter Not Currently Implemented
i.e. 10uF Cap Not Connected
- High Power to Low Power Connection Via DB9 Cable
U of A Robotics Laboratory
Title
PUMATORQ: PUMA Motor Current Measurement
Size
B
Document Number
MSC-1989-05
REV
1.0
Date
May 31, 1990
Sheet
1 of 1

Figure A.18 Current Measurement Circuitry

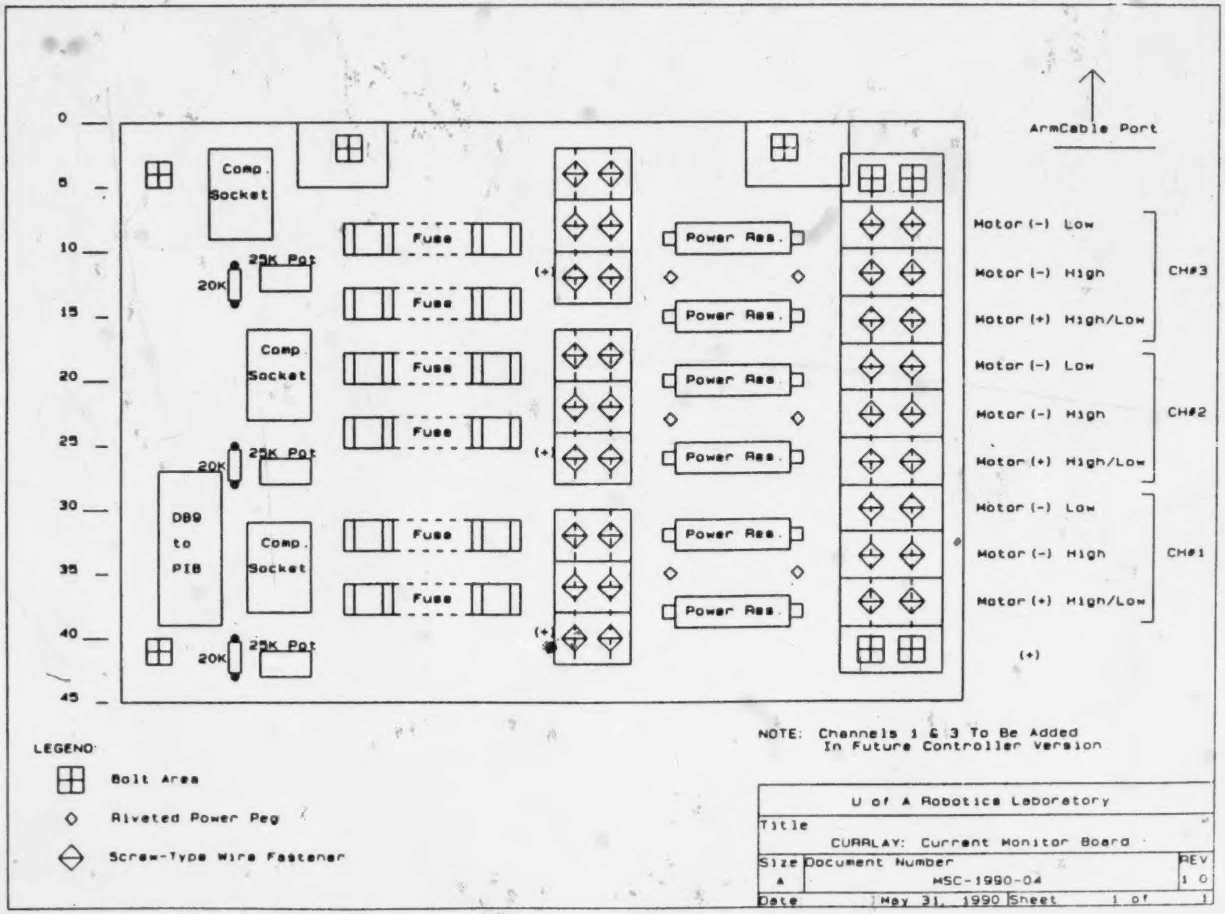

0
5
10
15
20
25
30
35
40
45
Comp. Socket
25K Pot
20K
Fuse
(+)
Power Res.
DB9 to PIB
ArmCable Port
Motor (-) Low
Motor (-) High
Motor (+) High/Low
CH#3
CH#2
CH#1
(+)
NOTE: Channels 1 & 3 To Be Added In Future Controller Version
LEGEND:
Bolt Area
Riveted Power Peg
Screw-Type Wire Fastener
U of A Robotics Laboratory
Title
CURRLAY: Current Monitor Board
Size
A
Document Number
MSC-1990-04
REV
1.0
Date
May 31, 1990
Sheet 1 of 1

Figure A.19 Current Sensing Board Layout

APPENDIX B - SOFTWARE REFERENCE AND LISTINGS

B.1 Software Reference

B.1.1 Power Up Sequence

To avoid equipment damage follow the power up sequence:

(1) TURN ON VMEbus chassis

(2) TURN ON the Unimate controller

This sequence is followed to prevent electrical transient damage to the DATEL DVME601B analog to digital converter in the VMEbus chassis. This board has only a 1500 ohm power off input impedance and hence a voltage spike during power up originating from the Unimate controller may cause system damage. When the DVME601B is powered up, its input impedance is 1000 Mohms.

Similarly to avoid equipment damage follow the power down sequence:

(1) TURN the Unimate Controller OFF

(2) TURN the VMEbus Chassis OFF

B.1.2 Software Start-Up Sequence

This section follows the stepwise process of loading the controller kernel into the TUTOR 68000, initializing the DVME601B 68010 analog to digital converter board, initializing and halting the Unimate controller and then beginning normal controller kernel program execution. Prior to beginning the process it is essential to read the Power up sequence given in section B.1.1.. After this sequence is understood and followed, the process of starting the system can be continued. A flowchart of the starting process is shown in Figure B.1.

To aid in the discussion a basic explanation of the TUTOR 68000 communication software is helpful. The TUTOR 68000 has two mode of operation. The first mode is the monitor level at which programs already downloaded into memory can be started and debugged using the monitor break point or trace features. This mode is run through the direct RS-232 link to the main terminal. The other mode is the TM or transparent terminal mode. This mode is used to talk to external devices which are connected to the second RS-232 line. For the controller system in its present configuration, this line is directly connected to serial port B of the SUN 3/160 computer; consequently, when the TM mode is used, the terminal connection is made directly to the SUN computer.

To begin the system start process turn the main system terminal on (QUME terminal) and wait for the TUTOR prompt. If this prompt does not appear check to make sure that channel A of the switch box is selected and press the RETURN key several times. If the TUTOR prompt is still not found,

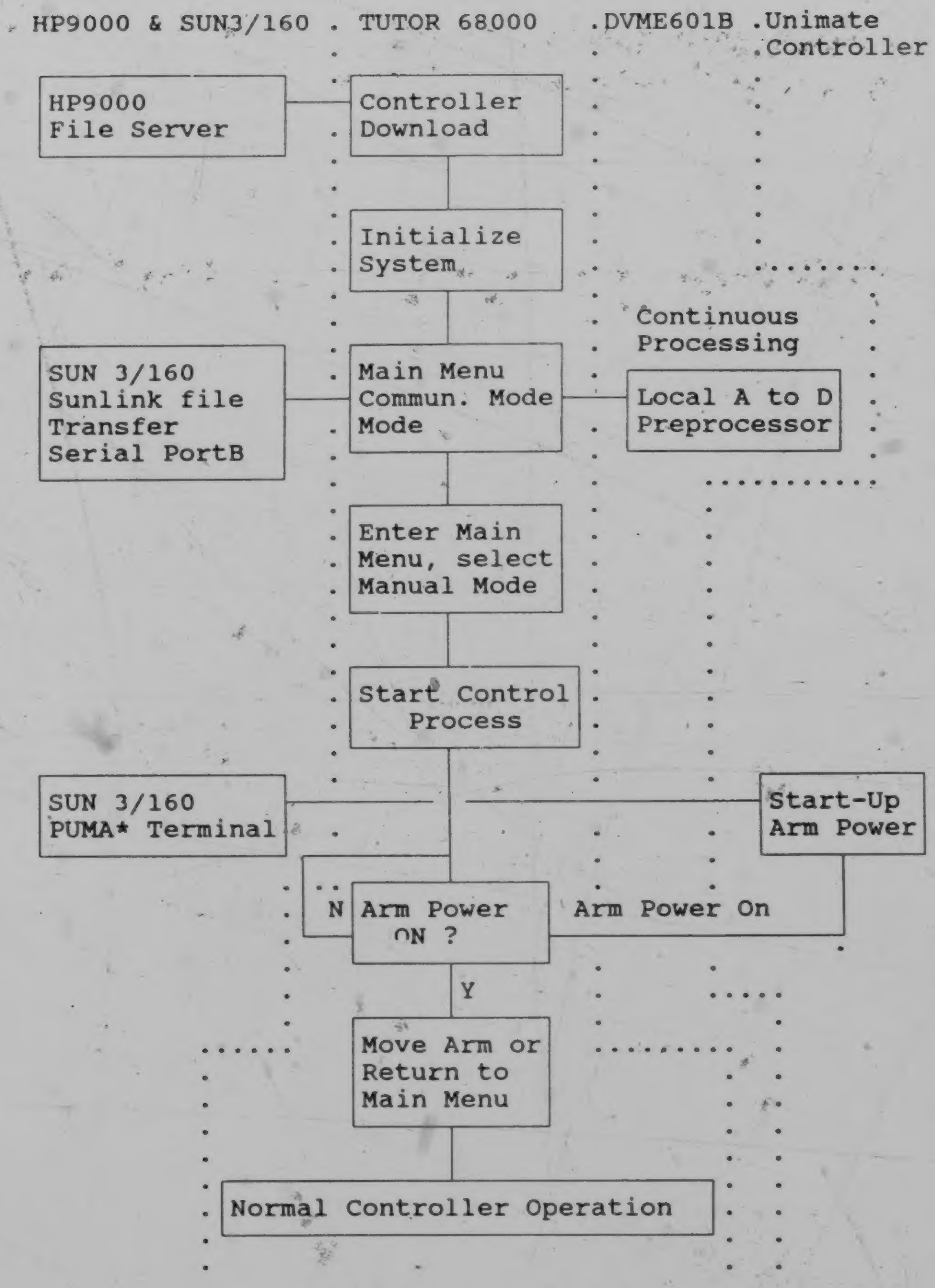

HP9000 & SUN3/160
TUTOR 68000
DVME601B
Unimate Controller
HP9000 File Server
Controller Download
Initialize System
Continuous Processing
SUN 3/160 Sunlink file Transfer Serial PortB
Main Menu Commun. Mode Mode
Local A to D Preprocessor
Enter Main Menu, select Manual Mode
Start Control Process
SUN 3/160 PUMA* Terminal
Start-Up Arm Power
N
Arm Power ON ?
Arm Power On
Y
Move Arm or Return to Main Menu
Normal Controller Operation

Figure B.1 Controller Software Start-Up

depress the TUTOR system reset switch. Once the TUTOR prompt appears enter the command "TM" to enter the transparent mode of the TUTOR monitor. Make sure the SUN 3/160 is running properly prior to this so that it is available to be accessed at this point. After typing "TM" followed by RETURN key, type the RETURN key several times until the SUN 3/160 UNIX prompt appears. Log on to the SUN UNIX system.

After successfully logging on the SUN UNIX system remote logon to the HP9000 UNIX system (The actual physical connection is done with an Ethernet cable run between the SUN 3/160 and the HP9000 system) since this is where the kernel and user programs have been developed. Move to the directory containing the kernel program and then return back to the TUTOR monitor level by typing CNTRL-A at the main terminal. Next type the command " LO ;X=cat control.h" to download the controller kernel program to the TUTOR 68000 board. A long download process of S-records will begin and the S-records will be echoed to the main terminal as they are downloaded. Note that the kernel program S-record file could have also been remote copied to the SUN 3/160 from the HP9000 machine in a previous step and the download of S-records performed directly from the SUN 3/160. In the method described above it is assumed that the kernel is being actively modified and recompiled on the HP9000. After the download is complete return to the transparent terminal mode by typing "TM".

Once in the transparent terminal mode, return to the SUN

3/160 machine by logging out from the HP9000. The SUN 3/160 contains two programs: (1) the DVME601B local assembly language program and (2) the SUNLINK communication program used to access the SUN file server. Return again to the transparent TUTOR mode and start the kernel program by typing the command "GO 1000". This command causes the TUTOR 68000 processor to jump to a kernel start routine (START in the skeleton.s assembly listing) and the kernel program runs automatically from this time forth.

The process mentioned above, starting from the download is shown below. Comments about each step are contained in { and } brackets. For a more detailed discussion of the modules called see the software section 4.0. User keyboard responses are shown in bold print.

```
{ Return to TUTOR level when logged on to the SUN 3/160 }

CNTRL-A    { Sequence typed to get to TUTOR level }

TUTOR  2.1 > LO ;X=control.h

{ A series of S-records is then echoed to the screen for
about four minutes}

S0030000FC
S113 ......
     .
     .
     .
S9030000FC

TUTOR  2.1 > TM

{ Hit carriage return several times }

{ Logout from the HP9000 and return to the TUTOR level }
```

```
bode: ~/current_directory> logout

robo: ~/robot_directory> ^A  (Type control-A)

TUTOR  2.1 > GO 1000
PHYSICAL ADDRESS=00001000
0000D902       ( Stack pointer and frame pointer
0000E902                 values, for diagnostics )

 Boot-up Initialization Begins
Global Channel Set
Choose Active Channel 1    ( Control which joint variables
Control Algorithm Time n msec     are being displayed )
############################################
Enter Main Menu of Program:
############################################
******************Main Controller Menu********************

 Select a number

1. Redisplay Main Menu
2. Calibrate Arm
3. Test Sensors
4. Manual Mode
5. User Mode
6. Change Active Channel (log purposes)
7. Communication Mode
8. Exit to TUTOR Monitor Level

ACTIVE CHANNEL--> 1 <--      ( A buzzing sound is heard )

==>7    (Select Mode to download DVME601B program )

( Start the sunlink.c program found on the SUN 3/160 in
  order to establish a file serving bridge between the
  SUN 3/160 and the TUTOR 68000 )

 Select Options
 1. Logfile Download (ENTER SUNLINK FIRST)
 2. 601B local program (ENTER SUNLINK FIRST)
 3. 601B command program
 't' Transparent terminal mode
 'q' for quit
 'x' EXIT SUNlink, reprint menu
 'r' START SUNlink

==> r
***Attempting to Execute SUNLINK***
Waiting for SUNLINK, Press cntrl-Q to stop
sunlink
```

```
   SUNLINK Bridge Program Executing    (sunlink.c says
                                         hello)

SUNLINK Operational!

( Once the sunlink.c program running on the SUN 3/160 is
  operational, communicate with it remotely to download
  the DVME601B local program to the 601B.  )

 Select Options
 1. Logfile Download (ENTER SUNLINK FIRST)
 2. 601B local program (ENTER SUNLINK FIRST)
 3. 601B command program
 't' Transparent terminal mode
 'q' for quit
 'x' EXIT SUNlink, reprint menu
 'r' START SUNlink

==> 2
FILENAME: program601b.h     (Echo local file name )
SUN Transmit Option Executing
Sending Filename
Waiting for SUN file server response to file open request

( periods are printed out to show that the process is still
active )

Transfer of S-records Begins ..............................

..........................................................

     (etc.....)
..............................

S-record Upload Successful

( Control is then returned to the communication menu.
  The 601B command program option is selected to start
  running the local program stored on the S-records on
  the DVME601B  )

 Select Options
 1. Logfile Download (ENTER SUNLINK FIRST)
 2. 601B local program (ENTER SUNLINK FIRST)
 3. 601B command program
 't' Transparent terminal mode
 'q' for quit
 'x' EXIT SUNlink, reprint menu
 'r' START SUNlink

==> 3
```

```
{ The DVME601B begins to execute the local program after
  a sequence of initialization and starting routines
  have been run.  This sequence runs and prints out a
  group of diagnostic messages }

 Attempting to turn 601B on
     .
     .
     .      (Multiple diagnostic messages are displayed)
 Waiting for HSTSTA601 to Host ACK
 Press 's' to continue            ( Only if things lock up)
 HSTSTA: 5041
 ADSTAT: ffff8400

 Select Options
 1. Logfile Download (ENTER SUNLINK FIRST)
 2. 601B local program (ENTER SUNLINK FIRST)
 3. 601B command program
 't' Transparent terminal mode
 'q' for quit
 'x' EXIT SUNlink, reprint menu
 'r' START SUNlink

==> x     ( Stop running sunlink.c on the SUN )

 Select Options
 1. Logfile Download (ENTER SUNLINK FIRST)
 2. 601B local program (ENTER SUNLINK FIRST)
 3. 601B command program
 't' Transparent terminal mode
 'q' for quit
 'x' EXIT SUNlink, reprint menu
 'r' START SUNlink

==> q     ( Return to the main controller menu )
*******************Main Controller Menu*****************

 Select a number

1. Redisplay Main Menu
2. Calibrate Arm
3. Test Sensors
4. Manual Mode
5. User Mode
6. Change Active Channel (log purposes)
7. Communication Mode
8. Exit to TUTOR Monitor Level

ACTIVE CHANNEL--> 1 <--

==> 4     (Select Mode to start robot arm power. )
```

{ At this point move over to the SUN 3/160 console, log in and run the PUMA* program. This program establishes a dumb terminal connection the Unimate controller through the RS-232 port A of the SUN }

{ Turn the Unimate controller on. The following prompts will appear on the SUN screen after some introductory messages. Type "n" for no to both of these prompts }

```
* VAL-II Boot B560.2.0 24AUG85 *

Load VAL-II from floppy (Y/N)? n



VAL-II 560.2.0G  P/N-935F31  H/W-O S/N-516 28FEB86  19Kw
COPYRIGHT 1981 - UNIMATION,  INC.
Initialize (Y/N)? n
     .             { VAL II prompt }
```

{ The Unimate controller will beep to indicate that it has successfully started }

{ Move back to the main terminal (QUME) and type 4 to enter the manual mode (see menu above) }

{ The manual mode prompt will then request that arm power be turned on. At this point press the ARM POWER ON button found on the front of the Unimate controller. Toggle the RUN/HALT switch to HALT then back to the RUN position. The RUN LED located on top of the Unimate controller for the LSI computer will then go out and the following information will be displayed on the SUN console which is still connected to the Unimate }

```
.
021370
@          {LSI-11 monitor prompt}
```

{ This prompt indicates that the Unimate LSI computer is halted. Type CONTROL-C at the SUN console to exit the PUMA* terminal program. The following message will appear, type y. After this log out from the console }

```
^C         { CONTROL-C }
 !!REALLY QUIT EMULATION MODE? (y or n): y

robo: ~\current_directory> logout
```

```
{ Return to the main terminal and type "c" at the manual
  mode prompt. }
**********************************************
Turn ARM ON!
Type c to continue once this is done
**********************************************


==> c
********MANUAL MODE EXECUTING********
   < - Large Move (-)          { move robot joint which is
   > - Large Move (+)            selected for the current
   [ - Small Move (-)            channel }
   ] - Small Move (+)
   s - stop robot arm & quit manual mode
   t - tune joint
   v - select which channel to view

{ At this point a continuous set of state variables for
  three joints is printed out.  The format is as follows:

  Vn   X.XXX en X sn X  Vn+1   X.XXX en+1 x sn+1 X...

  Vn identifies the view, i.e. the joint which is
  being viewed.  X.XXX is the voltage from the poten-
  tiometer of the selected joint.  "en X" is the
  count found on the encoder.  "sn X" is the set point
  which is currently being sent to the controller.
  Vn+1 is the variables for the next joint.     }

 V1  2.53 e1 0 s1 0 V2  2.5 e2 0 s2 0 V3  2.513 e3 1 s3 0
     .            .            .
     .            .            .   { runs continuously }
     .            .            .

{ The joint is moved in large or small increments using
  the <,>,[,] keys.  To change which joint is being
  moved type 'v' and the prompt to choose a channel will
  appear.  Type 6 for joint 6. }

v
 Choose Active Channel 6

{ At this point the variables for the upper joints 4,5
  and 6 will be displayed.  The message shown has the
  same format as the one for joints 1 to 3. }

V4   2.49 e4 1 s4 0 V5  2.89 e5 0 s5 0 V6  2.2 e6 0 s6 0


{ To exit the manual mode type "s" and the arm power
  will turn off.  A message will be displayed and the
```

```
{main menu will be printed }

STOPPED MANUAL MODE
*******************Main Controller Menu*******************

 Select a number

1. Redisplay Main Menu
2. Calibrate Arm
3. Test Sensors
4. Manual Mode
5. User Mode
6. Change Active Channel (log purposes)
7. Communication Mode
8. Exit to TUTOR Monitor Level

ACTIVE CHANNEL--> 6 <--

==>      {Note that channel (joint) 6 is now active. }
```

The controller is now started. The next step is to run the calibrate routine to calibrate the joint encoders relative to the home position. After this is done the user can proceed as desired for the program is now in its fully operational state. Typically the manual mode or user mode would be selected to move the arm for some experimental purpose. The communication mode might then be selected to download a logged data file to the SUN 3/160 or perhaps the system requires calibration and testing. The test sensor mode is described at the end of the hardware section.

B.1.3 Calibration and Testing of Boards

Calibration and testing of any of the boards used in this project can be done with a multimeter, a battery operated D.C. voltage source, an oscilloscope, and a copy of the board schematics, board layouts, and connector pin outs. The test sensor routine of the controller software is also used. This section describes calibration and testing methods for the encoder, PUMA interface and current sensor boards. The DATEL DVME 601B and DVME 628V boards are tested according to the instructions in their respective manuals.

The encoder board can be tested quickly and easily. The counter channels can be monitored directly by selecting the manual mode option found in the main menu of the controller program. This routine prints out the value of the encoder counter registers. By moving each robot joint individually and watching the encoder value, one can quickly see if the encoder is counting properly.

The PUMA interface board is tested by running the test sensor routine called from the main menu of the controller program and selecting the analog to digital converter monitoring option. After this has been done, the robot controller is turned on and the pot voltages coming from the robot arm, measured at the bus connector with the multimeter, are compared to the displayed values. The displayed voltage and multimeter voltages should agree quite closely (+/-10mV).

The remaining analog sensor readings, displayed by the analog to digital test sensor routine can also be used. Without the robot arm power turned on the sixth and eighth analog values shown by the test sensor routine should have a low or random voltage value. These are channels, reserved for joint number one and three current measurements, are not presently connected. The seventh analog value is connected to the joint number two current sensor and should read a D.C. offset value somewhere between 50 and 200 millivolts. This value should be recorded and used to correct motor current data if it is collected and analyzed. The ninth, tenth and eleventh analog values correspond to the inverted logic values of the SS RON HI, STOP L and BRK RLS MES PUMA status signals. Measure these signals with the multimeter where they enter the PIB bus connector. The high and low logic values of the measured voltages should correspond to the low and high values of the displayed analog values. Note that at the servo bus the SS RON HI signal is twelve volts or larger before being converted to TTL level by an optoisolator.

The motor servo and arm power enable signal commands which come from the DVME 628V digital to analog converter can be tested using the test sensor routine. Select the D to A Channel Output menu option and monitor the output of the differential servo motor opamps (refer to the DACBLK schematic in appendix A) with an oscilloscope. These outputs should be toggling between positive and negative ten volts. The same toggling signal should appear at the OFF L pin of the PIB servo bus edge connector (refer to the edge connector in the PUMAVME schematic found in Appendix A).

Finally, the current amplifier sensor board can be calibrated such that the current flowing to the PUMA robot joint two motor will equal the voltage measured at the analog to digital converter input of the DVME 601B. To perform this calibration, once again run the test sensor routine and monitor the seventh (joint two motor current voltage) analog to digital converter voltage. Remove the fuses found on the current sensor board and apply a small 50mV D.C. signal across the fuse holder clips on the voltage divider side of the sensor board. Next turn the Unimate controller on but do not load VAL II or do anything else. **Turning the robot arm power ON may SERIOUSLY OR FATALLY INJURE the testing individual or result in the robot arm dropping and being damaged! One should be aware of the differences between low power and high power Unimate chassis grounds. Understand what you are doing before making this measurement!** Observe the seventh analog value displayed by the test sensor routine. Adjust the one thousand ohm gain pot on the current amplifier circuit (located near the top edge of the PIB) until a voltage value of about 1.008 volts is obtained. The 1.008 volt reading is the same numeric value of the current that would be flowing through the resistor during robot arm operation,

i.e. Motor current = 50mV/0.0496 = 1.008 amps.

B.2 Program Listings

B.2.1 Control.c Listing

```
/**********************************************************
  control.c

    This program executes the functions necessary to run a
basic  PUMA 560 controller.  It is run on the TUTOR 68000
system  developed at the University of Alberta, Electrical
Engineering Dept..  It runs in a VMEbus based system which
includes  the  TUTOR  68000  as  the  real-time  supervisor
computer,  the  Unimate  controller,  and  the  SUN  3/160
workstation.
  The  hardware  used  in  the  real-time  system  consists  of
several modules:

  (1) AMline encoder - sets up AM3..5 lines
      - custom construction
  (2) Encoder board -monitors robot quadrature & index pulse
                     lines
      - custom construction
  (3) PIB - PUMA interface board between VME & Unimate
            systems
      - custom construction
  (4) CIB - Current interface board - monitors one of the
             JNT1..3 motor currents on the PUMA controller
      - custom construction
  (5) DVME601B - 16 channel A/D board from DATEL
      - monitors PUMA robot position pots, current signal,
        three system status/control lines
  (6) DVME628V - 8 channel D/A board from DATEL
      - sends servo signals to joints 1..6 & ARMOFF signal
        to servo amplifier circuitry.

  This program is menu driven.  All supported features are
available from a main menu.  User programs calling higher
level  robot  subroutines  must  be  compiled  separately  then
linked to this controller kernal.  The combined program is
then downloaded to the TUTOR 68000 system (or other target
Real-Time Processor).

  A character based terminal is used for the display.  The
PUMA VAL II controller is monitored using port A of the SUN
3/160 workstation.
**********************************************************/

/**********************************************************/
/*          External Assembler Functions                  */
/**********************************************************/
tern int encoder(void);
extern int rencoder(void);
extern dacontrol(void);
extern initial(void),clkunmask(void),clkmask(void);
extern armoff(void);
extern user();
/***********************************************************/
/*          Constant Declarations                          */
/***********************************************************/


/*......Miscellaneous Declarations......*/

#define ETX 003
#define ENQ 005
#define ACK 006

#define BUZZON 200
#define BUZZOFF 10000
#define BUZZTIMES 2

/* Define robot parameters */
#define JNTVELMAX 0.3 /* Maximum joint velocity rad/sec */
#define ANGRES1 0.00010045592  /*Define rads/encoder bit */
#define ANGRES2 0.00007285144
#define ANGRES3 0.00011710810
#define ANGRES4 0.00008266054
#define ANGRES5 0.00008737555
#define ANGRES6 0.00008188683

/* Maximum deviation allowed from control algorithm  */
#define DVMAX 4000


/*......Encoder Board Declarations......*/

/*  Encoder Board Declarations */
#define STATUS_ENC ((short int*)0x02000C)
#define UNMASK_ENC ((short int*)0x02000E)
#define INDLOAD_ENC ((short int*)0x020000)
#define LATCH_ENC ((short int*)0x020012)

/* Define controller traj. step size in encoder positions */
#define RAMPSTEP 5

/* Number of encoder points error allowed to meet a
   trajectory objective. */
#define ENCTOL50 /* Number of encoder points */


/*......TUTOR 68000/68681 Duart Declarations......*/

/*  Device dependent output addresses for TUTOR board  */
/*  These values must be changed for the DVME601B board */
/*  Also the registers for the DVME601B are different
    than those found on the TUTOR board
    The character routines are found in the DMVE601B ROM */

#define TERM_STATA ((char *)0xEFFC03)
#define TERM_CMDA  ((char *)0xEFFC05)
#define TERM_RXTXA ((char *)0xEFFC07)
#define TERM_STATB ((char *)0xEFFC13)
#define TERM_CMDB  ((char *)0xEFFC15)
#define TERM_RXTXB ((char *)0xEFFC17)


/*......DVME601B Declarations......*/

/* 601B Reserved Locations */

   /*   Cmnd intrpt to DVME-601B  */
#define CMDINT_601B ((short int *)0xFAFFFC)

   /*   VMEbus intrpt ID reg (r/w) */
#define INTREG_601B ((short int *)0xFAFFFA)

   /*   Enbl/Dsbl VMEbus intrpt bit7 */
#define INTENB_601B ((short int *)0xFAFFF8)

   /*   Exec HSTSTA word (r/w)   */
#define HSTSTA_601B ((short int *)0xFAFFF6)

   /*   Exec ADSTAT word (r/w)   */
#define ADSTAT_601B ((short int *)0xFAFFF4)

   /*   Beginning of channels 0..15  */
#define CHANNEL_601B ((short int *)0xFA0000)

   /*   Local 601B status word  */
#define STATWD_601B ((short int *)0xFA0010)

   /*   Mode for local 601B  */
#define MODEWD_601B ((short int *)0xFA0012)


/* Define the top and bottom of DPR RAM */
#define DPRHI_601B ((char *)0xFAFFFF)
#define DPRAFB_601B ((char *)0xFAFFEF)
#define DPRLO_601B ((char *)0xFA0000)


/*......DVME628V Declarations......*/

/* Define DAC safety limits in volts */
#define DACMAX5
#define DACMIN-5
/*  Set the maximum and minimum servo signals allowed  */
#define DACLO -16000
```

```
#define DACHI 16000

/***********************************************************/
/*         Global Variables                               */
/***********************************************************/

int rampstp;/* Size of trajectory step */
int activech;/* Active channeld to monitor   */

/* Encoder board count for joint i */
short int enc[7];
short int basenc[7];   /* Encoder equiv. of 0 degrees */

/* Controller set point & goal set point position */
short int sppos[7],newsppos[7];


/*  PD,PID controller variables */
float kp[7],kd[7],ki[7];
float kidelta[7],kddelta[7];
int kpf[7],kideltaf[7],kddeltaf[7]; /*fixed point coeffs. */
int contrl[7];/* controller output signal */
short int conbin[7];/*      "       "      "  to DAC */
int errpos[7],olderr[7],iterm[7],isum[7]; /* Control var.*/
float pstep,istep,dstep;    /* Gain adjustment variables */
                            /* for tuning of gains  */

/* Timing variables*/
int  sampletime;  /* controller sample period in msec  */
float ticktime,delta; /* tick-base time div.;
                         delta-control div. */
short int logclock;     /* clock in msec for log file    */
int realclock;
int ctick;   /* control system clock*/

/* SYSTEM FLAGS*/
int clockflag; /* =1 clock runs, =0 clock stops */
intcontrolflag;     /* Flag to detect control algorithm
                        finished within sample time */
int conoff; /*Flag to detect control algor. turned on =0, */


/* SAFETY Limits*/
  /* Maximum encoder values allowed */
short int encmax[7],encmin[7];

  /* Maximum deviation from control law */
short int devmax;

  /* Maximum pot range allowed      */
short int potmax[7],potmin[7];
```

```
/* KEYBOARD monitoring function */
int inputready,keytick;
char iochar,bufchar;


/* LOGGING FUNCTION*/
   /* RAM buffer pointers and storage for logging*/
short int logindx;
short int logpnt[4000];

   /* Logging flags*/
short int loggo;       /* Log function ON */
short int enclog,spposlog,conlog,currlog; /* Log data sel.*/
   /* Counter between log points*/
short int logtick,logperiod;


/* ENCODER BOARD*/
   /* Global encoder variables, for option selection */
char enc_mask,enc_command,echannel;


/* DVME601B Related Variables*/

   /* DPR pointers for AFB & S-Record download */
char *dpraddrc,*dpraddrs;

   /* Executive control variable*/
short int execerr;

/***********************************************************
  main
    This function initializes system variables then prompts
the user with a main menu from which branches can be made to
all subroutines of the program.

***********************************************************/
main()
{

int i,menusel,xit;

/* Initialize system variables */
bootup();

xit=0;
menusel=0;
mainmenu();    /* Print main menu */
while(xit!=1)
{
   menusel=fgetc();   /* Poll keyboard for user resp.*/
```

```
    switch(menusel)
    {
      case '1':   /* Redisplay main menu */
           mainmenu();
           break;
      case '2':   /* Calibrate robot arm */
           rampstp=1;
           reqarmon();
           calibrate();
           rampstp=RAMPSTEP;
           break;
      case '3':   /* Calibrate sensors   */
           testsensor();
           mainmenu();
           break;
      case '4':   /* Run Manual Mode    */
           /* Request robot arm power be turned on   */
           reqarmon();
           manualmode();
           mainmenu();
           break;
      case '5':   /* Run a User Program */
           /* Request robot arm power be turned on   */
           reqarmon();
           user();
           mainmenu();
           break;
      case '6':   /* Change viewed channel */
           i=chpick();
           activech=i; /* Set global chan.pointer*/
           mainmenu();
           break;
      case '7':   /* Connect to SUN computer */
           comlink();
           mainmenu();
           break;
      case '8':  /* Quit the program, goto TUTOR */
           xit=1;
           break;
    }
}

clkmask();  /* Turn robot & IRQ7 interrupt clock off*/
conoff=0;
untick();

stopthis:
buzz();/* Wake up the user */

/************************************************************
   RETURN to TUTOR Monitor
```

```
**********************************************************/
}

/**********************************************************
   mainmenu
      This function prints out the main controller menu.

**********************************************************/
mainmenu()
{

printf("\015\n**********MAIN CONTROLLER MENU************");
printf("\015\n\015\n Select a number\015\n ");
printf("\015\n1.Redisplay Main Menu ");
printf("\015\n2.Calibrate Arm ");
printf("\015\n3.Test Sensors  ");
printf("\015\n4.Manual Mode   ");
printf("\015\n5.User Mode     ");
printf("\015\n6.Change Active Channel (log purposes)");
printf("\015\n7.Communication Mode ");
printf("\015\n8.Exit to TUTOR Monitor Level ");
printf("\015\n\015\nACTIVE CHANNEL--> %d <--",activech);
prompt();
}

/**********************************************************
   bootup
     This function does basic system initialization and
checking of device hardware.

**********************************************************/
bootup()
{

int i;

printf("\015\n Boot-up Initialization Begins ");

/* Call assembly routine to lock arm off, set up AMcode,
   initialize system clock and encoder board       */
initial();

/* Set controller variables to default values */
defset();

loginit();   /* Start logfile to log selected vars.*/

activech=1;  /* Def. channel to be actively monitored */

/* Choose controller channel to monitor first */
printf("\015\nGlobal Channel Set");
activech=chpick();
```

```
/* Set safety limit on control law deviation */
devmax=DVMAX;

/* Set safety limits on pot deviation */
/* Based on measurements taken 890328 LK */
/* 10% rounded value (All values in volts ) */
potmax[1]=3.2; potmin[1]=1.4;
potmax[2]=3.8; potmin[2]=0.8;
potmax[3]=3.1; potmin[3]=1.7;
potmax[4]=3.6; potmin[4]=1.5;
potmax[5]=3.0; potmin[5]=1.8;
potmax[6]=4.3; potmin[6]=0.2;


/*
  multiply values by 3276.8 to get 2's complement binary
  for the A/D use.*/

i=1;
while (i<7)
{
   potmax[i]=potmax[i]*3276.8;
   potmin[i]=potmin[i]*3276.8;
   i=i+1;
}

rampstp=RAMPSTEP;    /* Set the trajectory increment */

/* Precalculate some PID coefficients & initialize
   fixed point PID coefficients                       */
coeffcalc('i');

controlspeed();  /* See control algorithm exec.time */

printf("\015\n######################################");
printf("\015\nEnter Main Menu of Program:");
printf("\015\n######################################");

conoff=1;  /* Disable control alg. till user/man modes */
buzz();    /* Alert the user that system has started */

}

/******************************************************
   manualmode
       This function allows the user to run the robot joints
manually.

NOTE: To get a step response, change rampstp=2000 so that the
trajectory controlling function will try to move 2000 encoder
points instantly.
******************************************************/
```

```
manualmode()
{

int i,j,logalerted;
char choice;

printf("\015\n********MANUAL MODE EXECUTING********");
printf("\015\n   < - Large Move (-)");
printf("\015\n   > - Large Move (+)");
printf("\015\n   [ - Small Move (-)");
printf("\015\n   ] - Small Move (+)");
printf("\015\n   s - stop robot arm & quit manual mode");
printf("\015\n   t - tune joint  ");
printf("\015\n   v - select which channel to view ");
i=activech;  /* Display active global channel */
choice = 'c';
tick();/* Start realtime clock              */
logalerted=1;
j=0;

while((choice!='s')&&(conoff==0))
{
     choice = getc();
     switch(choice)
     {
     case '.':  /* Large Move (-) */
          printf("\015\n negative");
          /* Move joint by positive encoder positions */
          logon();   /* Log transient response */
          logalerted=0;
          /*  Move +2000 encoder points */
          newsppos[i]=newsppos[i]+2000;
          break;

     case ',':  /* Large Move (+) */
          printf("\015\n positive");
          /* Move joint by negative encoder positions */
          logon();
          logalerted=0;
          /*  Move -2000 encoder points */
          newsppos[i]=newsppos[i]-2000;
          break;

     /*  case 0x27:  Medium Move, future feature?
          newsppos[i]=newsppos[i]+;
          break;
     case ';':
          newsppos[i]=newsppos[i]-;
          break;
     */
     case ']':  /* Small Move (-) */
          printf("\015\n negative");
```

```
            /* Move joint by positive encoder positions */
            logon();   /* Log transient response */
            logalerted=0;
            /*  For Nichols Pert. use 100 pulse  */
            newsppos[i]=newsppos[i]+100;
            break;

        case '[':  /* Small Move (+) */
            printf("\015\n negative");
            /* Move joint by positive encoder positions */
            logon();   /* Log transient response */
            logalerted=0;
            /*  For Nichols Pert. use 100 pulse  */
            newsppos[i]=newsppos[i]-100;
            break;

        case 's':  /* Stop Manual mode */
            printf("\015\n STOPPED MANUAL MODE");
            break;

        case 't':  /* Tune active joint controller */
            /* Tune joint of users choice */
            tunepid();

            /* Update controller variables */
            coeffcalc('i');
            break;

        case 'v':  /* Sel. active channel */
            i=chpick();
            activech=i;   /* set global channel pointer */
            break;
        }

    /* Detect when logging done, tell user */
    if ((loggo==0)&&(logalerted!=1))
    {
       logalerted=1;
       printf("\015\nLogging Completed");
    }

    /* Select which joint to print out values for */
    if (i>3)
       j=1;
    else
       j=0;

    switch(j)
    {
    /*  Change subscripts back to i later */
        case 0:  /* Display CH1..3 */
            printf("\015\nv1%8.3f",(*(CHANNEL_601B)/3276.8));
```

```
          printf(" e1 %d",enc[1]);
          printf(" s1 %d",sppos[1]);
          printf(" v2%8.3f",(*(CHANNEL_601B+1)/3276.8));
          printf(" e2 %d",enc[2]);
          printf(" s2 %d",sppos[2]);
          printf(" v3%8.3f",(*(CHANNEL_601B+2)/3276.8));
          printf(" e3 %d",enc[3]);
          printf(" s3 %d",sppos[3]);
          break;

    case 1:  /* Display CH4..6  */

          printf("\015\nv4%8.3f",(*(CHANNEL_601B+3)/3276.8));
          printf(" e4 %d",enc[4]);
          printf(" s4 %d",sppos[4]);
          printf(" v5%8.3f",(*(CHANNEL_601B+4)/3276.8));
          printf(" e5 %d",enc[5]);
          printf(" s5 %d",sppos[5]);
          printf(" v6%8.3f",(*(CHANNEL_601B+5)/3276.8));
          printf(" e6 %d",enc[6]);
          printf(" s6 %d",sppos[6]);
          break;
     }

}

logoff();  /* Stop logging function      */
clkmask(); /* Turn off periodic IRQ7 interrupts */
conoff=1;  /* Stop calls to control algorithm */
}

/*********************************************************
   reqarmon
      This function requests that the user turn the robot arm
power on.  Before allowing the user to do this the controller
is enabled and the handler call to the controller is enabled.

*********************************************************/
reqarmon()
{

int i;
clkunmask();     /* Make sure controller is running */
conoff=0;

/* Provide bumpless transfer when arm turns on.
   This may have implications for some trajectory
   following routines.*/

rencoder();
       i=1;
while (i<7)
```

```
{
   sppos[i]=enc[i];
   i=i+1;
}

armon();/* Allow robot arm to be turned on */
printf("\015\n **************************************");
printf("\015\n Turn ARM ON!");
printf("\015\n Type c to continue once this is done");
printf("\015\n **************************************");
prompt();
while(getc()!='c') {}
tick();   /* Turn real-time clock on to be used for
     ramped trajectory motion*/
}

/**********************************************************
  controlspeed
     Compute the single cycle time of the control algorithm

**********************************************************/
controlspeed()
{

conoff=1;    /* Turn off all but clock */
        /* Calculate controller performance in msec       */
tick();    /* Prepare to count real-time intervals */
clkunmask(); /* Turn on periodic interrupts */
control(); /* Measure control algorithm execution time */

untick();
printf("\015\nControl Algorithm Time %d msec",realclock);

clkmask();
conoff=0;   /* reassign to default   */
}

/**********************************************************
   loginit
      Set up the default variables to be logged.

**********************************************************/
loginit()
{

/* Set up which values to put in log file*/
/* Store these values in the first 8 bytes of
   log buffer*/
logindx=0;

/* Encoder Logging */
enclog=1;
```

```
logpnt[logindx]=enclog;
logindx=logindx+1;
/* Setpoint Position Logging*/
spposlog=1;
logpnt[logindx]=spposlog;
logindx=logindx+1;
/* Control Variable Logging*/
conlog=1;
logpnt[logindx]=conlog;
logindx=logindx+1;
/* Current Logging JNT#2*/
currlog=1;
logpnt[logindx]=currlog;
logindx=logindx+1;

/* Do not start logging until told*/
loggo=0;

/* Set time between logging of points in msec   */
logtick=0;
logperiod=sampletime;

/* Initialize elapsed log period clock*/
logclock=0;
}

/*************************************************************
  prompt
     Print default controller prompt to terminal

*************************************************************/
prompt()
{

printf("\015\n");
printf("\015\n==>");
}

/*************************************************************
  defset
     Set controller variables to default values

*************************************************************/
defset()
{

int i;

/*  Set Interrupt keyboard monitoring variables */
inputready=0;
keytick=0;/* Count to next keyboard sample */
iochar=0x00; bufchar=0x00;
```

```
/* Initialize control variables */
i=1;
while (i<7)
{
   olderr[i]=0;
   isum[i]=0;
   i=i+1;
}

/* Turn the call to the control algorithm off */
conoff=1;

/* Set base clock rate*/
ticktime=.001;
/* Set sample rate*/
ctick=0;
sampletime=3;/* sample time in msec*/
delta=sampletime/1000.0; /* control int. delta time */

controlflag = 0;/* controller not active */

/* Get the zero degrees encoder value          */
/*    Clear the encoder */
enc_command = 'c';
encoder();
enc_command = 'r';
encoder();

/* Set controller setpoints to hold arm steady */
i=1;
while (i<7)
{
   basenc[i]=0.0;
   sppos[i]=enc[i];
   newsppos[i]=enc[i];
   i=i+1;
}

/* Set default KI of PID controller  */
ki[1]=0.0; ki[2]=0.0; ki[3]=0.0;
ki[4]=0.0; ki[5]=0.0; ki[6]=0.0;

/*
   "Compensation of Unmodeled Puma Manipulator Dynamics",
   IEEE 1987 Conference on Robotics & Automation, Vol.1,
   M.B. Leahy, Jr, and G.N. Saridis.

PD Controller:
          Pos. Gain      Vel. Gain
Joint 1   225            30
Joint 2   400            40
Joint 3   225            30
Joint 4   900          60
Joint 5   900          60
Joint 6   1225         70

PID Controller:
          Pos.Gain     Vel. Gain      Int. Gain
Joint 1   300          30             1000
Joint 2   300          30             1000
Joint 3   300          30             1000
Joint 4   700          50             3000
Joint 5   700          50             3000
Joint 6   900          60             4000

   */

/* NOTE: a fixed point PD controller is used   */
/* Gains lower than above, slower sample time ?*/
kp[1]=153; kp[2]=275; kp[3]=448;
kp[4]=422; kp[5]=448; kp[6]=448;

kd[1]=5; kd[2]=10; kd[3]=11;
kd[4]=10; kd[5]=10; kd[6]=10;

/* Set PID controller tuning increments*/
pstep=1; istep=1; dstep=1;

/* Set the software limits on joint encoders */
/* Based on observations of PUMA maximum values */
/* Refer to lab book notes 890618
  (Round to thousand(Total counts/rotation))-2000 */
encmax[1]=25000; encmin[1]=-25000;
encmax[2]=27000; encmin[2]=-27000;
encmax[3]=18000; encmin[3]=-18000;
encmax[4]=29000; encmin[4]=-29000;
encmax[5]=17000; encmin[5]=-17000;
encmax[6]=26000; encmin[6]=-26000;

/* Initialize Dual Port RAM BUFFER */
/* Useful in detecting that the 601B is not updating
   its buffer correctly (value=-10V)*/
i=0;
while (i<12)
{
   *(CHANNEL_601B+i)=0x8000;
   i=i+1;
}

}

/**************************************************************
  chpick
     This function picks the channel to display and returns
```

```
it to the calling function.

************************************************************/
chpick()
{

int i,j;
printf("\015\n Choose Active Channel");
j=0;
while(j!=1)
{
    i=getc();
                                                i           f
((i=='1' ||(i=='2')||(i=='3')||(i=='4')||(i=='5')||(i=='6'))
         j=1;
}

i=i-48;/* Convert ASCII to a numeric index */
return(i);
}

/*************************************************************
   tunepid
     This function adjusts the controller coefficients under
user control.  FOR BEST RESULTS use with coeffcalc.

   Command Summary:
   ================
   "," - reduce P
   "." - increase P
   ";" - reduce I
   "'" - increase I
   "[" - reduce D
   "]" - increase D

************************************************************/
tunepid()
{

int i;
char choice;
printf("\015\n Tune Controller ");
i=activech;       /* Tune actively monitored channel */
choice = 'c';
while(choice!='q')
{
  choice = getc();
  switch(choice)
  {
     case ',':  /* KP tuning (-)*/
          printf("\015\n negative");
          kp[i]=kp[i]-pstep;
          break;
     case '.':  /* KP tuning (+)*/
          printf("\015\n positive");
          kp[i]=kp[i]+pstep;
          break;
     case ';':  /* KI tuning (-)*/
          printf("\015\n negative");
          ki[i]=ki[i]-istep;
          break;
     case 0x27: /* KI tuning (+)*/
          printf("\015\n positive");
          ki[i]=ki[i]+istep;
          break;
     case '[':  /* KD tuning (-)*/
          kd[i]=kd[i]-dstep;
          break;
     case ']':  /* KD tuning (+)*/
          kd[i]=kd[i]+dstep;
          break;
     case 'q':
          printf("\015\n quit tuning");
          break;
  }
  printf("\015\n kp= %f ki= %f kd= %f",kp[i],ki[i],kd[i]);
}
}

/************************************************************
  coeffcalc
    This function reduces the number of calculations by
premultiplying certain terms.  It also calculate the
equivalent PID fixed point coefficient values.

************************************************************/
coeffcalc(command)
char command;
{

kidelta[1]=ki[1]*delta/2; kidelta[2]=ki[2]*delta/2;
kidelta[3]=ki[3]*delta/2; kidelta[4]=ki[4]*delta/2;
kidelta[5]=ki[5]*delta/2; kidelta[6]=ki[6]*delta/2;

kddelta[1]=kd[1]/delta; kddelta[2]=kd[2]/delta;
kddelta[3]=kd[3]/delta; kddelta[4]=kd[4]/delta;
kddelta[5]=kd[5]/delta; kddelta[6]=kd[6]/delta;

if (command=='i')  /* Integer trunc. for PID controller */
{
     /* Set fixed point equivalent of gain variables */
     kpf[1]=kp[1]; kpf[2]=kp[2]; kpf[3]=kp[3];
     kpf[4]=kp[4]; kpf[5]=kp[5]; kpf[6]=kp[6];
```

```
        kideltaf[1]=kidelta[1]; kideltaf[2]=kidelta[2];
        kideltaf[3]=kidelta[3]; kideltaf[4]=kidelta[4];
        kideltaf[5]=kidelta[5]; kideltaf[6]=kidelta[6];

        kddeltaf[1]=kddelta[1]; kddeltaf[2]=kddelta[2];
        kddeltaf[3]=kddelta[3]; kddeltaf[4]=kddelta[4];
        kddeltaf[5]=kddelta[5]; kddeltaf[6]=kddelta[6];

}
else
        {} /* Running fixed point controller */
}

/***************************************************************
   handlerc:
      This routine serves as a 6-joint PID controller for the
PUMA.  It is called by the assembler program "HANDLER" which
performs the necessary low-level interrupt hardware reset and
RTE functions which are not available at the "C" level.

***************************************************************/
handlerc()
{

/******** Realtime clock counter function **************/
if (clockflag==1) {realclock = realclock+1;}

if (realclock > 10000000) {realclock=0;}

/******************* LOGGING FUNCTION ******************/
if (logindx>=3000)
   logoff(); /* Stop logging when buffer full */
if ((loggo==1)&&(conoff==0))
{
   logtick=logtick+1;
   if (logtick==logperiod) \
   {
      logtick=0;
      logclock=logclock+logperiod;
      /* Begin loading the log buffer      */
      logpnt[logindx]=logclock;
      logindx++; /* logclock is 2 bytes long*/
      if (enclog==1)
      {
         logpnt[logindx]=enc[activech];
         logindx++;
      }

      if (spposlog==1)
      {
         logpnt[logindx]=sppos[activech];
         logindx++;
      }

      if (conlog==1)
      {
         logpnt[logindx]=conbin[activech];
         logindx++;
      }

      if (currlog==1)
      {
         /* Obtain current reading CH7, JNT#2 */
         logpnt[logindx]=*(CHANNEL_601B+7);
         logindx++;
      }
   }
}   /* End of logging function*/

/*IMPLEMENT WITH HIGHER SPEED CPU
if (((*CHANNEL_601B)&&(0x0007))!=0)
{
   conoff=1;
   clkmask();
   logoff();
printf("\015\n EXECUTION TERMINATES AS PUMA SIGNALS LOST ");
   */
   /* Tell user what caused abnormal termination  */
   /*
   if (((*CHANNEL_601B)&&(0x0001))==1)
      printf("\015\n Amplifier or other Problem ");
   if (((*CHANNEL_601B)&&(0x0002))==1)
      printf("\015\n Arm Power OFF ");
   if (((*CHANNEL_601B)&&(0x0004))==1)
      printf("\015\n Manual Brake Release Engaged ");
}
*/

/********* Joint Movement Section **********************/
/* Alternates with keyboard input check   */
if ((conoff==0)&&(realclock>=2))
{
   realclock=0;
   /* Move joints as set points change                */
   if (newsppos[1]<sppos[1])
   {
      sppos[1]=sppos[1]-rampstp;
      if (newsppos[1]>sppos[1])
          sppos[1]=newsppos[1];
   }
   else
   {
      if (newsppos[1]>sppos[1])
         sppos[1]=sppos[1]+rampstp;
   if (newsppos[1]<sppos[1])
      sppos[1]=newsppos[1];
}

if (newsppos[2]<sppos[2])
{
   sppos[2]=sppos[2]-rampstp;
   if (newsppos[2]>sppos[2])
      sppos[2]=newsppos[2];
}
else
{
   if (newsppos[2]>sppos[2])
      sppos[2]=sppos[2]+rampstp;
   if (newsppos[2]<sppos[2])
      sppos[2]=newsppos[2];
}

if (newsppos[3]<sppos[3])
{
   sppos[3]=sppos[3]-rampstp;
   if (newsppos[3]>sppos[3])
      sppos[3]=newsppos[3];
}
else
{
   if (newsppos[3]>sppos[3])
      sppos[3]=sppos[3]+rampstp;
   if (newsppos[3]<sppos[3])
      sppos[3]=newsppos[3];
}

if (newsppos[4]<sppos[4])
{
   sppos[4]=sppos[4]-rampstp;
   if (newsppos[4]>sppos[4])
      sppos[4]=newsppos[4];
}
else
{
   if (newsppos[4]>sppos[4])
      sppos[4]=sppos[4]+rampstp;
   if (newsppos[3]<sppos[4])
      sppos[4]=newsppos[4];
}

if (newsppos[5]<sppos[5])
{
   sppos[5]=sppos[5]-rampstp;
   if (newsppos[5]>sppos[5])
      sppos[5]=newsppos[5];
}
```

```
   else
   {
      if (newsppos[5]>sppos[5])
         sppos[5]=sppos[5]+rampstp;
      if (newsppos[5]<sppos[5])
         sppos[5]=newsppos[5];
   }

   if (newsppos[6]<sppos[6])
   {
      sppos[6]=sppos[6]-rampstp;
      if (newsppos[6]>sppos[6])
         sppos[6]=newsppos[6];
   }
   else
   {
      if (newsppos[6]>sppos[6])
         sppos[6]=sppos[6]+rampstp;
      if (newsppos[6]<sppos[6])
         sppos[6]=newsppos[6];
   }

}
else
{
   /* Buffered keyboard input */
   /* Get keyboard input about every 20 msec */
   keytick=keytick+1;
   if (keytick==10)
   {
      keytick=0;
      iochar=fgetc();/* Poll to see if key pressed */
      if (iochar!=0x00)
      {
         bufchar=iochar;    /* Buffer input character */
         inputready=1;
      }
   }
}


/************** CONTROLLER FUNCTION *********************/
/* Count clock cycles before controller run */
if (conoff==0)
{
   ctick=ctick+1;

   if (ctick==sampletime)  /* control period ? */
   {
     if ((ctick==sampletime)&&(controlflag==1))
     /*Overrun, control algorithm too slow! */
     {
        clkmask();  /* Stop periodic clock & robot */
        conoff=1;
        logoff();
        printf("\015\n Control algorithm too slow!!");
        printf("\n\n\n\n");
     }
     else
     {
        ctick = 0;
        controlflag = 1;/* control algorithm exec.*/
        control();
        controlflag = 0;    /* control algorithm done  */
     }
   }
}
}

/**********************************************************
  control:
    This routine is a PID algorithm which controls the robot
  joint angles.

**********************************************************/
control()
{

/* Store all encoder counter values */
rencoder();

/* Compare encoder values with required setpoint */
errpos[1]=sppos[1]-enc[1]; errpos[2]=sppos[2]-enc[2];
errpos[3]=sppos[3]-enc[3]; errpos[4]=sppos[4]-enc[4];
errpos[5]=sppos[5]-enc[5]; errpos[6]=sppos[6]-enc[6];

/*  Stop arm if controller fails to keep arm within
     a user defined trajectory error.              */
if (abs(errpos[1])>devmax)
{
   clkmask();/* stop timer */
   conoff=1;
   logoff();
   printf("\nJoint#1 control error");
   goto end;
}
if (abs(errpos[2])>devmax)
{
   clkmask();/* stop timer */
   conoff=1;
   logoff();
   printf("\nJoint#2 control error");
   goto end;
}
```

```
if (abs(errpos[3])>devmax)
{
   clkmask();/* stop timer */
   conoff=1;
   logoff();
   printf("\nJoint#3 control error");
   goto end;
}
if (abs(errpos[4])>devmax)
{
   clkmask();/* stop timer */
   conoff=1;
   logoff();
   printf("\nJoint#4 control error");
   goto end;
}
if (abs(errpos[5])>devmax)
{
   clkmask();/* stop timer */
   conoff=1;
   logoff();
   printf("\nJoint#5 control error");
   goto end;
}
if (abs(errpos[6])>devmax)
{
   clkmask();/* stop timer */
   conoff=1;
   logoff();
   printf("\nJoint#6 control error");
   goto end;
}

/* PD control*/
/* P=KP*error signal*/
/* D=KD*(deltaerror)/deltatime          */
contrl[1]=kpf[1]*errpos[1]+kddeltaf[1]*(errpos[1]-olderr[1]);
contrl[2]=kpf[2]*errpos[2]+kddeltaf[2]*(errpos[2]-olderr[2]);
contrl[3]=kpf[3]*errpos[3]+kddeltaf[3]*(errpos[3]-olderr[3]);
contrl[4]=kpf[4]*errpos[4]+kddeltaf[4]*(errpos[4]-olderr[4]);
contrl[5]=kpf[5]*errpos[5]+kddeltaf[5]*(errpos[5]-olderr[5]);
contrl[6]=kpf[6]*errpos[6]+kddeltaf[6]*(errpos[6]-olderr[6]);

/* Check for under/overflow, control value out of range */
if (contrl[1]<DACLO) {contrl[1]=DACLO;}
else
   if (contrl[1]>DACHI) {contrl[1]=DACHI;}

if (contrl[2]<DACLO) {contrl[2]=DACLO;}
else
   if (contrl[2]>DACHI) {contrl[2]=DACHI;}
if (contrl[3]<DACLO) {contrl[3]=DACLO;}
else
   if (contrl[3]>DACHI) {contrl[3]=DACHI;}

if (contrl[4]<DACLO) {contrl[4]=DACLO;}
else
   if (contrl[4]>DACHI) {contrl[4]=DACHI;}

if (contrl[5]<DACLO) {contrl[5]=DACLO;}
else
   if (contrl[5]>DACHI) {contrl[5]=DACHI;}

if (contrl[6]<DACLO) {contrl[6]=DACLO;}
else
   if (contrl[6]>DACHI) {contrl[6]=DACHI;}


/* Control signal in 2's complement binary */
/* Change control sign to match plant actuation */
/* For a floating point controller this is where
   contrl (which would be of type float) would be
   converted to 2's complement binary.  */
conbin[1] = -contrl[1];
conbin[2] = -contrl[2];
conbin[3] = -contrl[3];
conbin[4] = -contrl[4];
conbin[5] = -contrl[5];
conbin[6] = -contrl[6];

/* Send the control signal to the D to A channels.  */
/* To preserve speed, no arguments are used.  The   */
/* DAC values are found in the global variables:    */
/*  conbin1..conbin6      */
/* This is where the assembly language routine takes*/
/* them from.      */

dacontrol();

/* compute error of n-1 for next iteration */
olderr[1]=errpos[1];   olderr[2]=errpos[2];
olderr[3]=errpos[3];
olderr[4]=errpos[4];   olderr[5]=errpos[5];
olderr[6]=errpos[6];
end: ;
}  /* El finito!*/


/**********************************************************
  USER LIBRARY

  Contents:
  SINGLE JOINT:
```

```
  (1) movee(jnt,enc)-move by the specified encoder count from
                     the present location
  (2) movea(jnt,enc)-move by the specified angle
  (3) moveta(jnt,enc)-move to specified absolute angle

  MULTIPLE JOINTS:
  (1) vectorje(enc1,...,enc6)
  (2) vectorja(ang1,...,ang6)   FUTURE addition
  (3) vectorc(x,y,z,o,a,t)   FUTURE addition

***********************************************************/

/**********************************************************
  movee
    This function changes the set point for the required
joint to the new position if possible, then waits until it is
within the encoder tolerance signifying the set point has
been met.
This routine requires an encoder count increment value.
  i.e. how many encoder points the robot must move +/-

***********************************************************/
movee(jnt,encincr)
int jnt;
short int encincr;
{

short int encval;
if ((jnt>0)&&(jnt<7))
{
   /* Check to see if requested increment moves the
      arm joint out of its operating envelop      */
   encval=newsppos[jnt];
   encval=encval+encincr;
   /* Yes, reduce it so that it just goes to boundary */
   if (encval>encmax[jnt])
   {
      encval=encmax[jnt];
      printf("\015\n Joint Increment Movement Reduced");
   }
   if (encval<encmin[jnt])
   {
      encval=encmin[jnt];
      printf("\015\n Joint Increment Movement Reduced");
   }
   /* Request required incremental move */
   newsppos[jnt]=encval;
}
else
   printf("\015\n Invalid Joint Specification in movee()");

printf("\015\n Joint %d Moves",jnt);
```

```
/* Wait until joint moves to within tolerance of
   selected encoder count.*/
while (abs(enc[jnt]-newsppos[jnt])>ENCTOL)
{
     /*  Do nothing but move arm to end point */
}
printf("\015\n Joint %d Completes Movement",jnt);

}

/**************************************************************
  movea
    This function changes the set point for the required
joint to the new position if possible, then waits until it is
within the encoder tolerance signifying the set point has
been met.
This routine requires an angular degree increment value.
  i.e. how many degrees to move from current position

**************************************************************/
movea(jnt,angle)
int jnt;
double angle;
{

short int encval;

angle=angle*(3.141592654)/180.0;  /* Convert to rads */
if ((jnt>0)&&(jnt<7))
{
   /* Calculate encoder count equivalent of angle */
   switch(jnt)
   {
          case 1:
               encval=angle/ANGRES1;
               break;
          case 2:
               encval=angle/ANGRES2;
               break;
          case 3:
               encval=angle/ANGRES3;
               break;
          case 4:
               encval=angle/ANGRES4;
               break;
          case 5:
               encval=angle/ANGRES5;
               break;
          case 6:
               encval=angle/ANGRES6;
               break;
   }
movee(jnt,encval);
}
else
  printf("\015\n Invalid Joint Specification in movea()");
}

/************************************************************
   moveta
     This function moves the joint requested to the given
  absolute angle position.  The angle position requested is
in degrees.

************************************************************/
moveta(jnt,angle)
int jnt;
double angle;
{

double currangle,angres;

if ((jnt>0)&&(jnt<7))
{
   /* Identify angular resolution of selected joint */
   switch(jnt)
   {
     case 1: angres=ANGRES1; break;
     case 2: angres=ANGRES2; break;
     case 3: angres=ANGRES3; break;
     case 4: angres=ANGRES4; break;
     case 5: angres=ANGRES5; break;
     case 6: angres=ANGRES6; break;
   }
   /* Calculate present angular position */
   if (jnt==2)
   {
      /* Angle starts at -90 degrees for this joint */
      /* This is based on Craig's derivation of
         kinematics and inverse kinematics     */

currangle=((enc[jnt]-basenc[jnt])*angres)*180.0/3.1415926;
    currangle=currangle-90;
   }
   else
   {
   /* Assumption that the base encoder value is zero */

currangle=((enc[jnt]-basenc[jnt])*angres)*180.0/3.1415926;
   }

   /* Move by an increment to require angle */
   movea(jnt,angle-currangle);
}
```

```
else
   printf("\015\n Invalid Joint Specification in movee()");
}

/*************************************************************
   vectorje
      This function servos all of the joints simultaneously
by the required encoder increment.
*************************************************************/
vectorje(j1,j2,j3,j4,j5,j6)
int j1,j2,j3,j4,j5,j6;
{

short int encval;
int i,flg1,flg2,flg3,flg4,flg5,flg6,stp,inc[7];
int e1,e2,e3,e4,e5,e6;

inc[1]=j1; inc[2]=j2; inc[3]=j3;
inc[4]=j4; inc[5]=j5; inc[6]=j6;

/* Initialize stopping flags */
flg1=0; flg2=0; flg3=0; flg4=0; flg5=0; flg6=0;

/* User input to stop execution of current vector*/
stp=0;

i=1;

/* Check to see if increment exceeds encoder count */
while (i<7)
{
   encval=newsppos[i];
   encval=encval+inc[i];
   if (encval>encmax[i])
   {
      encval=encmax[i];
      printf("\015\n Joint Increment Movement Reduced");
   }
   if (encval<encmin[i])
   {
      encval=encmin[i];
      printf("\015\n Joint Increment Movement Reduced");
   }
   /* Request required incremental move  */
   newsppos[i]=encval;
   i=i+1;
}
printf("\015\n Type 't' to Terminate Current Movement");

while (stp!=1)
{
   /* Stop movement when done */
   if ((flg1==1) && (flg2==1) && (flg3==1) && (flg4==1) &&
      (flg5==1)&&(flg6==1))
      stp=1;
   /* Check for User intervention to stop trajectory */
   if (getc()=='t')
      stp=1;
   /* Check to see if joints have moved to within
      the specified trajectory tolerance*/
   e1=abs(enc[1]-newsppos[1]);
   e2=abs(enc[2]-newsppos[2]);
   e3=abs(enc[3]-newsppos[3]);
   e4=abs(enc[4]-newsppos[4]);
   e5=abs(enc[5]-newsppos[5]);
   e6=abs(enc[6]-newsppos[6]);

   /* Flag tolerances met as they occur */
   if (e1<ENCTOL)
      flg1=1;
   if (e2<ENCTOL)
      flg2=1;
   if (e3<ENCTOL)
      flg3=1;
   if (e4<ENCTOL)
      flg4=1;
   if (e5<ENCTOL)
      flg5=1;
   if (e6<ENCTOL)
      flg6=1;
 printf("\015\nERRORS %d %d %d %d %d %d",e1,e2,e3,e4,e5,e6);
}
}

/**********************************************************
  FUNCTION LIBRARY
**********************************************************/

/**********************************************************
   tick:
Turns the real-time clock on.  Clock is initialized
to zero.

**********************************************************/
tick()
{

realclock = 0;
clockflag = 1;
}

/**********************************************************
   untick:
Turns the real-time clock off.  The clock count value
```

```
   and the equivalent value in seconds are printed out if the
   function parameter is set to -1.

***********************************************************/
untick(p)
int p;
{

clockflag = 0;
if (p==-1)
{
   printf("\n\nElapsed Clock:\n");
 printf("Count %d :Seconds%f",realclock,realclock*ticktime);
}
}

/***********************************************************
   logon:
Turns the logging clock on.  It also resets the logfile
pointer to the beginning of the buffer and initializes the
logclock to zero.  The first 8 bytes of the log buffer
contain the flags which indicate the type of data stored at
each clock cycle.

***********************************************************/
logon()
{

loggo=1; logclock=0;
logindx=4;   /* Go past the data flags */
}

/***********************************************************
   logoff:
Turns the logging clock off.

***********************************************************/
logoff()
{

loggo=0;
}

/***********************************************************
  fputc:
  This routine sends a character to a USER-DEFINED DUART
  terminal hardware device.  This represents the TUTOR
  terminal in this case.

***********************************************************/
int fputc(chr,device)
int chr;
```

```
void *device;
{

char statusreg;/* DUART status register */
char charout;   /* Output as an unsigned char. */
statusreg = 0;
charout=chr;

if ((*TERM_STATA)&(0x10)==16)    /* Overrun Error */
{
     return(-1);
}
else
{
     while(statusreg!=4)/* ready to transmit */
     {
          statusreg = (*TERM_STATA)&(0x04);
     }

     *TERM_RXTXA = charout;  /* Send char. to terminal */
     return(chr);
}

}

/************************************************************
  rputc: (remote put character)
     This routine sends a character to a USER-DEFINED DUART
terminal hardware device.  This routine is for communication
to remote processors.

************************************************************/
int rputc(chr)
int chr;
{

char statusreg; /* DUART status register */
char charout;   /* Output as an unsigned char. */
int     stattemp;
statusreg = 0;
charout=chr;

stattemp=*TERM_STATB;
if((stattemp)&(0x10)==16)    /* Overrun Error */
{
     *TERM_CMDB=0x40;   /* Reset error bits */
     *TERM_CMDB=0x30;
     *TERM_CMDB=0x04;
     return(-1);

}
else
{
    while(statusreg!=4) /* ready to transmit? */
    {
         statusreg = (*TERM_STATB)&(0x04);
    }

     *TERM_RXTXB = charout; /* Send character */
}
}

/**********************************************************
  rgetc:
This routine gets a character from the remote RS-232 source
if a character has been entered there. It does this by
checking the status register.  This routine returns a NULL
character (0x00) if there is an overrun or other error
problem.  It returns a NUL character (0x00) if no keyboard
input has been found.

***********************************************************
rgetc()
{

char statusreg,chararrived;

statusreg = (*TERM_STATB)&(0xF1);
chararrived = (statusreg)&(0x01);

/* Check to see if error bits 3-7 set */
if (statusreg >= 7)
{
   *TERM_CMDB = 0x40;  /* reset status bits after error*/
   *TERM_CMDB = 0x20;
   *TERM_CMDB = 0x01;  /* Enable receiver */
   return(0x00);    /* Return NULL if error */
}
/* Check if a character has arrived */
else
   if (chararrived == 1)
      return((*TERM_RXTXB)&(0x7F));
   else /* no character found */
      return(0x00);
}

/**********************************************************
  fgetc:
This routine gets a character from the keyboard if a
character has been entered there.  It does this by checking
the status register.  This routine returns a NUL character
(0x00) if there is an overrun or other error problem.  It
returns a NUL character (0x00) if no keyboard input has been
found.
```

```
**********************************************************/
fgetc()
{

char statusreg,chararrived;

statusreg = (*TERM_STATA)&(0xF1);
chararrived = (statusreg)&(0x01);

/* Check to see if error bits 3-7 set */
if (statusreg >= 7)
{
   *TERM_CMDA = 0x40;  /* reset status bits after error*/
   return(0x00);    /* Return NULL if error */
}
/* Check if a character has arrived */
else
{
   if (chararrived==1)
      return(*TERM_RXTXA);
   else
      return(0x00);/* no character found */
}
}

/**********************************************************
  getc
    This function checks the interrupt character buffer
(bufchar) to see if a keyboard character has been pressed.
(inputready=1) It is equivalent to fgetc() if the system
clock is not running.
**********************************************************/
getc()
{

char retval;

retval=0x00;     /* Default NULL character */

if (conoff==1)
{
   retval=fgetc();    /* No interrupts running, poll  */
   /* hardware directly      */
}
else
{
  if (inputready==1) /* Check for buffered key chars.*/
  {
     inputready=0;
     retval=bufchar;
  }
}
```

```
return(retval);
}

/*************************************************************
  buzz - this routine sends a buzzing sound to the terminal
  every time it is called.  It is to be a warning to the user
  that some system condition has changed.

*************************************************************/
buzz()
{

int buzzcount,buzzrep;
int dud;

buzzcount = 0;
buzzrep = 0;

/* Count buzz interval      */
while((++buzzrep)!=BUZZTIMES)
{
  buzzcount = 0;
  /* Send BUZZON BEL characters  */
  while((++buzzcount)!=BUZZON)
  {  dud=fputc(0x07,0);}

  buzzcount = 0;
  /* Stop sending for BUZZOFF count */
  while((++buzzcount)!=BUZZOFF)
  {  /* Do nothing */ }
}
}

/*************************************************************
  comlink
This function starts up the SUNLINK program found on the
SUNUNIX system and then prompts the user to choose from the
available file transfer options.  The following options are
supported:
(1) Transfer of data file from the VME system to the SUN
(2) Transfer of 601B local program "s" record file from
the SUN system to the TUTOR VME to the 601B DPR.
(3) Transfer of 601B command start up file from the SUN
system to the TUTOR VME to the 601B DPR.

*************************************************************/
comlink()
{

int  rperr,cmdsel,miscexit,sunactive,fileflag,chrcount;
short int logbuf;
charchrbuf[20],ch,ch2;
```

```
sunactive=0;

/* Print the menu of options*/
commenu();

while((cmdsel=fgetc())!='q')
{
   switch(cmdsel)
   {
     case '1':
          /* Download Logfile to SUN*/
          lineget(10,&chrcount,&chrbuf[0]);

          /* Add file extension for selected option */
          chrbuf[chrcount]='.';
          chrbuf[chrcount+1]='v';
          chrbuf[chrcount+2]='1';
          chrcount=chrcount+2;

          /* Echo filename in chrbuf[i]    */
          fileecho(chrcount,&chrbuf[0]);
          fileflag=0x06;

          /* General send program:*/
          fileflag=fileopen('s',chrcount,&chrbuf[0]);
          if ((fileflag==0x15)||(fileflag!=0x06))
          {
               printf("\015\nDownload file open invalid");
               /* Perform no further action */
          }
          else
          {
               printf("\015\nBegin Logfile Download\015\n");
               /* Transfer Data from buffer to SUN file */
               miscexit=0;
               logbuf=0;
               while((miscexit!=1)&&(logbuf<logindx))
               {
               /* Send the ASCII equiv. of data to SUN */
                    miscexit=sendasc(logbuf,'i',2);
                    logbuf=logbuf+1;
               }

             /* Signal the end of transmission */
                 while((rperr=rputc(0x03))==-1)   {}
          }
          printf("\015\nDownload to SUN complete");
          commenu();
          break;

     case '2':
          /* Upload 601B local program from SUN*/
     chrcount=0;
     chrbuf[chrcount++]='p'; chrbuf[chrcount++]='r';
     chrbuf[chrcount++]='o'; chrbuf[chrcount++]='g';
     chrbuf[chrcount++]='r'; chrbuf[chrcount++]='a';
     chrbuf[chrcount++]='m'; chrbuf[chrcount++]='6';
     chrbuf[chrcount++]='0'; chrbuf[chrcount++]='1';
     chrbuf[chrcount++]='b'; chrbuf[chrcount++]='.';
     chrbuf[chrcount]='h';

     /* Echo filename in chrbuf[i]   */
     fileecho(chrcount,&chrbuf[0]);

     fileflag=0x06;
     /* General send program:*/
     fileflag=fileopen('t',chrcount,&chrbuf[0]);
     if ((fileflag==0x15)||(fileflag!=0x06))
     {
        printf("\015\nUpload file open invalid");
        /* Perform no further action */
     }
     else
     {
     /* Transfer Data from buffer to SUN file */
        printf("\015\n Transfer of S-Records Begins ");
        miscexit=0;
        dprsrecord('i',0x00);
        while((miscexit!=-1)&&(miscexit!=ETX))
        {
          /* Receive S-record characters from SUN */
          miscexit=getasc();
          printf(".");
          if ((miscexit!=-1)&&(miscexit!=ETX))
          {
            /* Load character into DPR memory */
            dprsrecord('l',miscexit);
          }
          else
            if (miscexit==-1)
               printf("\015\nS-record Upload Error");
        }
     }

     if (miscexit!=-1)
        printf("\015\nS-record Upload Successful");
     commenu();
     break;

case '3':
     /* Transfer 601B command program to 601B
        This program is stored in memory.     */

     if (execon()==0)
```

```
          {
             /* Load initialization AFB into DPR */
             printf("\015\n Starting INITIAL AFB ");
             dprinitafb();
             afbexecut('i');  /* run initialization */

             /* Load continuous AFB into DPR  */
             printf("\015\n Starting CONTINUOUS AFB ");
             dprcontafb();
             afbexecut('c');   /* run continuous AFB */
          }

          commenu();
          break;

     case 't':
          /* Start transparent terminal mode */
          printf("\015\n Transparent Terminal Mode:");
          printf("\015\n Type cntrl-a to exit      ");
          ch2=fgetc();
          while (ch2!=0x01) /* check for cntrl-a */
          {
             ch=rgetc();
             /* Display characters returned by SUN */
             if (ch!=0x00)
                fputc(ch,0);

             /* Scan keyboard to get user's keystrokes */
             ch2=fgetc();
             if ((ch2!=0x00)&&(ch2!=0x01))
                rputc(ch2);  /* Echo to SUN */
          }

          commenu();
          break;

     case 'q':
          /* Exit communication program*/
          break;

     case 'x':
          /* Stop SUNLINK execution on the SUN*/
          sunoff();
          commenu();
          break;

     case 'r':
          /* Start SUNLINK execution on the SUN*/
          sunon();
          commenu();
          break;
     }
```

```
}
}

/***********************************************************
  commenu
    This routine prints the communication subroutine menu.

***********************************************************/
commenu()
{

   printf("\015\n");
   printf("\015\n Select Options:");
   printf("\015\n 1.Logfile Download (ENTER SUNLINK FIRST)");
printf("\015\n 2.601B local program (ENTER SUNLINK FIRST)");
   printf("\015\n 3.601B command program");
   printf("\015\n 't' Transparent terminal mode ");
   printf("\015\n 'q' for quit");
   printf("\015\n 'x' EXIT SUNlink, reprint menu");
   printf("\015\n 'r' START SUNlink");
   prompt();
}

/***********************************************************
   fileopen
      Send the bridge command to the SUN and send the
filename to be stored or retrieved.

***********************************************************/
fileopen(command,chrcount,chrbuf)
int chrcount;
char *chrbuf,command;
{

intindx;
char ch;

rputc(005);    /* Send ENQ character to start SUN command
                  mode                            */

rputc(command);  /* Confirm option selected */

ch=rgetc();
/* Confirm option selected */
while(ch!=0x06)
{
   printf("%c",ch);  /* Make sure receivefile selected */
   ch=rgetc();
}

printf("\015\nSending Filename");
indx=0;
```

```
while(indx<=chrcount)
{
   rputc(chrbuf[indx]);
   indx=indx+1;
}
printf("\015\nWaiting for SUN file server response");
printf(" to file open request (fileopen)");
printf("\015\n");
rputc(005);    /* Send out enquire */

ch=rgetc();    /* Wait for SUN response */
while(ch==0)
{
   ch=rgetc();
}

return(ch);
}

/************************************************************
  fileecho
    This function echos the filename to the primary terminal.

************************************************************/
fileecho(chrcount,chrbuf)
int chrcount;
char *chrbuf;
{

int i;

printf("\015\nFILENAME: ");
i=0;/* Echo the file to be accessed  */
while(i<=chrcount)
{
   printf("%c",chrbuf[i]);
   i=i+1;
}
printf("\015\n");
}

/************************************************************
  sunon:
    This program starts the SUNLINK program.

************************************************************/
sunon()
{

char ch;
printf("\015\n***Attempting to Execute SUNLINK***");
rputc('s');
rputc('u');
rputc('n');
rputc('l');
rputc('i');
rputc('n');
rputc('k');
rputc(0x0D);
rputc(0x0A);
printf("\015\nWaiting for SUNLINK, Press cntrl-Q to stop");
printf("\015\n");
ch=rgetc();

/* Print SUNLINK welcome & wait for an ACK character */
while((ch!=0x06)&&(ch!=0x11))
{
   ch=rgetc();
   if (ch!=0x00)
      printf("%c",ch);    /* Print SUNLINK Welcome   */
}
if (ch==0x06)
    printf("\015\nSUNLINK Operational!");
printf("\015\n");
}

/***********************************************************
  wait
    This function sleeps until the USER types 's'.  This is
a useful debugging tool to step through a routine one stage
at a time.

***********************************************************/
wait(s)
char s;
{

while(rgetc()!=s)
   printf("Z");     /* Sleeping ZZZZ..... */
}

/***********************************************************
  sunoff
    This program turns the SUNLINK program off.

***********************************************************/
sunoff()
{

rputc(0x18);   /* Send CAN character */
}

/***********************************************************
  lineget
```

This subroutine acts as a simple line editor which acquires a string of specified maximum length and returns this string and its size to the calling routine.

```
lineget(chrmax,chrcount,chrbuf);

chrmax - maximum number of characters in the character string
chrcount - number of characters entered prior to the
           terminating carriage return character
chrbuf - 0..n character long character array

***********************************************************/
lineget(chrmax,chrcount,chrbuf)
intchrmax,*chrcount;
char*chrbuf;
{

int bufferdone,dud;
int retval;
char ch;

bufferdone=0;
retval=0;
*chrcount=0;

/* Prompt for user to supply the filename */
printf("\015\nFILENAME==>");
while(bufferdone!=1)
{
     ch=fgetc();/* poll for user input */
     if(ch!=0x00)
     {
       if (ch==0x0D)  /* carriage return after filename */
       {
          if (*chrcount==0)
          {
               retval=-1;  /* no filename entered, error */
               bufferdone=1;
          }
       else
       {
          /* Skip to next line & exit editor */
          retval=0;
          bufferdone=1;  /* valid filename entered */
       }
     }
     else
     {
       if (ch=='.')
          dud=fputc(0x07,0); /*BEEP, not allowed */
       else
       {
          if (ch==0x08)  /* Backspace check */
          {
            if (*chrcount==0)
               dud=fputc(0x07,0);/*BEEP, no characters! */
            else  /* Backspace, remove one character */
            {
               *chrcount=*chrcount-1;
               dud=fputc(0x08,0); /* Backspace */
               dud=fputc(0x20,0); /* Space character */
               dud=fputc(0x08,0); /* Backspace */
            }
          }
          else  /* valid character found */
          {
            if (*chrcount>=chrmax)
               dud=fputc(0x07,0); /*too many characters*/
            else
            {
               /* Add a character to the buffer*/
               *(chrbuf+*chrcount)=ch;
               *chrcount=*chrcount+1;
               dud=fputc(ch,0); /* Echo char. to console*/
            }
          }
       }
     }
}
}
return(retval); /*0:buffer filled okay, -1 :buffer empty */
}

/*************************************************************
   sendasc:
      This function sends the ASCII equivalent of a number to
a file which was previously opened on the SUN.  If an
unrecoverable error occurs during transmission the function
returns 1.  If execution occurred without error then the
function returns 0.

   parameters:

   logbuf - pointer to memory buffer.
   dattype - type of data type represented by buffer value at
             current pointer.
   numbyte - number of bytes in the selected data type.

*************************************************************/
sendasc(logbuf,dattype,numbyte)
int logbuf;
chardattype;
intnumbyte;   /* Number of bytes per data element */
{
```

```
int rperr,retval,conint,strgindx,digitcnt;
short int conshort;
charascstrg[20],conchar;
doubleconflt;

retval=0;

ascstrg[0]=32; /* First position equals a space */
if (dattype=='i')
{
   if (numbyte==2)  /* short integer data elements */
   {
      conshort=logpnt[logbuf];
      if (conshort<0)
      {
           ascstrg[0]=0x2D; /* place '-' in string */
           conshort=-conshort;
      }

      /* convert number to floating point */
      conflt=conshort;
   }
   else  /* number of bytes =4 */
   {
      conint=logpnt[logbuf];
      if (conint<0)
      {
           ascstrg[0]=0x2D;  /* place '-' in string */
           conint=-conint;
      }
      /* convert number to floating point */
      conflt=conint;
   }

   /* Count the number of integer digits */
   digitcnt=0;
   if ((conflt<1)&&(conflt>=0))
      digitcnt=1;
   else
   {
      while ((conflt>=1)&&(digitcnt<15))
      {
         conflt=conflt/10;
         digitcnt=digitcnt+1;
      }
   }
   /* Extract the digits one by one and
      and convert to ASCII                */
   strgindx=0;
   while (digitcnt!=0)
   {
      conflt=conflt*10;
      /* Correct round off error */
      conflt=conflt+0.0000001;
      conint=conflt;
      conchar=conint;    /* Extract base 10 digit */
      conchar=conchar+48;   /* Convert char. to ASCII */
      conflt=conflt-conint;  /* drop integer part */

      /* Store result in a string */
      ascstrg[strgindx+1]=conchar;
      strgindx=strgindx+1;
      digitcnt=digitcnt-1;
   }
   /* Add '+' sign to represent a LF */
   strgindx=strgindx+1;
   ascstrg[strgindx]='+';
   strgindx=strgindx+1;

   retval=0;
   while ((digitcnt!=strgindx)&&(retval!=1))
   {
      conchar=ascstrg[digitcnt];
      /* Wait to clear transmit error condition */
      rperr=rputc(ascstrg[digitcnt]);
      while(rperr==-1)
      {
          /* send a digit of the number to the SUN */
           rperr=rputc(ascstrg[digitcnt]);
      }
      digitcnt=digitcnt+1;
   }
}
else
{
   if (dattype=='f')
   {
      printf("\015\nNo floating point data convert!");
      retval=1;    /* Not implemented */
   }
}
return(retval);
}

/*********************************************************
   getasc
      This function gets an ASCII character from a file and
   returns it to the calling function.  If an error
   occurs during transmission "-1" is returned by the
    function.

*********************************************************/
g[illegible]tasc()
{
```

```
char ch;
char ch2;
int rperr;
int retval;

retval=0;
while((ch=rgetc())==0) {}  /* Wait for character */
if (ch!=ETX)
{

   /* Echo character to SUN */
   while((rperr=rputc(ch))==-1)  {}
   /* Wait to see if this was acknowledged or rejected */
   while((ch2=rgetc())==0) {}
   if (ch2!=ACK)
   {
      printf("\015\nNO ACK!");
      /* retry getting character if error */
      while (ch2!=ACK)
      {

         while((ch=rgetc())==0) {}
         while((rperr=rputc(ch))==-1)  {}
         while((ch2=rgetc())==0) {}
      }
      /* Too many retries error, STOP */
      if (ch2==0x15)
           retval=-1;
      while((ch2=rgetc())==0) {}
   }

   /* ELSE ACK, do nothing successful transmission */
}
if (retval==-1)
   return(retval);
else
{
   /* Convert coded LF & CR back to 0x0A & 0x0D */
   if (ch==0x0C)
      return(0x0A);
   else
   {
      if (ch==0x0B)
         return(0x0D);
      else
         return(ch);
   }
}

}

/**********************************************************
```

```
  dprcommand
     This function writes the 601B command file (application
function block in DATEL terms) to the high part of DPR
memory.  The function writes from highest reserved AFB memory
downward ($FAFFEF-$FA0000).  It also has an initialization
mode which resets the DPR pointer.

***********************************************************/
dprcommand(command,ch)
char command,ch;
{

/*  Decrement the supplied address (supplied as a pointer)
    after each call of the function
    */
/* Skip any CR or LF characters for DATEL format */
if ((command!=0x0D)&&(command!=0x0A))
{
   if (command=='l')  /* Load memory mode */
   {
      *dpraddrc=ch;
      dpraddrc=dpraddrc-1;
    }
    else
      if (command=='i')  /* Initialize memory pointer */
         dpraddrc=DPRAFB_601B;
    }
}

/***********************************************************
   execon
     This function turns the 601B executive ON.
  If an error occurs the function returns -1,
  otherwise 0 is returned.
***********************************************************/
execon()
{

printf("\015\n Attempting to turn 601B on");

asm( "  LEA.L$FAFFE6,A0  " );
asm( "  MOVE.B#'E',(A0)+" );
asm( "  MOVE.B#'X',(A0)+" );
asm( "  MOVE.B#'E',(A0)+" );
asm( "  MOVE.B#'C',(A0)+" );
asm( "  MOVE.B#' ',(A0)+" );
asm( "  MOVE.B#'O',(A0)+" );
asm( "  MOVE.B#'N',(A0)+" );
asm( "  MOVE.B#$0D,(A0)" );

/* Execute EXEC ON command */
return(startcom());
```

```
}

/************************************************************
  dprinitafb
     This function downloads the AFB & Function control
  codes which are used to initialize the 601B.

************************************************************/
dprinitafb()
{

char i,code[50];
code[0]='0'; code[1]='G'; code[2]='E'; code[3]='B';
/* Set up transfer of data to local buffer          */
code[4]=0x58; code[5]=0x01; code[6]=0x00; code[7]=0x00;
/* Call routine to start delay sampling timer */
code[8]=0x00; code[9]=0x90; code[10]=0x02; code[11]=0x00;
code[12]='F'; code[13]='D'; code[14]='N'; code[15]='E';
code[16]='L'; code[17]='D'; code[18]='P'; code[19]='D';
code[20]='S'; code[21]='R'; code[22]='T'; code[23]='S';
code[24]='0'; code[25]='L'; code[26]='E'; code[27]='S';
code[28]='C'; code[29]='D'; code[30]='N'; code[31]='E';

/* Initialize DPR pointer */
dprcommand('i',' ');

/* Transfer AFB to DPR  */
i=0;
while (i<32)
{
   /* Load AFB & command file to memory */
   dprcommand('l',code[i]);
   i=i+1;
}
}

/************************************************************
  dprcontafb
     This function downloads the AFB & Function control
  codes which are used to run the 601B in continuous mode.

************************************************************/
dprcontafb()
{

char i,code[50];
code[0]='0'; code[1]='G'; code[2]='E'; code[3]='B';
/* TRRDSC mode */
code[4]=0xA0; code[5]=0x01; code[6]=0x00; code[7]=0x00;
code[8]=0x00; code[9]=0x00; code[10]=0x00; code[11]=0x00;
code[12]=0x0B; code[13]=0x00; code[14]=0x00; code[15]=0x00;
```

```
/* Call preprocessing routine */
code[16]=0x00; code[17]=0x80; code[18]=0x02; code[19]=0x00;
code[20]='F'; code[21]='D'; code[22]='N'; code[23]='E';
code[24]='T'; code[25]='N'; code[26]='O'; code[27]='C';
code[28]='C'; code[29]='D'; code[30]='N'; code[31]='E';

/* Initialize DPR pointer */
dprcommand('i',' ');
i=0;
while (i<32)
{
   /* Load AFB & command file to memory */
   dprcommand('l',code[i]);
   i=i+1;
}

}

/***********************************************************
   afbexecut
      This function starts an afbblock running.  It runs the
initialize AFB block if 'i' is supplied as an argument and
waits for its completion.  It starts the continuous AFB
sampling block if 'c'is supplied and exits, not waiting for
it to finish.(obviously!)

***********************************************************/
afbexecut(command)
charcommand;
{

int fncterr,retval;

retval=0;
/* Start initialization AFB execution */
if (command=='i')
{
   startcom();  /* Start AFB and function command */
   while ((((*HSTSTA_601B)&(0x01))!=1)&&(command!='s'))
   {
      printf("\015\n Waiting for HSTSTA 601 to Host ACK ");
      if (getc()=='s')
      {
         command='s';
      }
   }
   fncterr=0;
   /* Test for AFB0 done yet or for error condition
   fncterr!=0x40   */
   while ((((*HSTSTA_601B)&(0x0040))!=0x0040)&&(fncterr==0))
   {
      printf("\015\n Waiting for AFB0 execution ");
      /* Check for Function or other error */
      fncterr=(*HSTSTA_601B)&(0x003E);
   }
   if (fncterr==0)   /* function 0 done */
   {
      printf("\015\n Initialization Successful. ");
   }
   else
   {
      printf("\015\n Function 0 Initialization Error.");
      retval=-1;
   }
}
else  /* Start continuous 601B execution  */
{
   startcom();  /* Start AFB & function commands */
   printf("\015\n Pausing for AFB0 Continuous Run ");
   printf("\015\n Waiting for HSTSTA 601 to Host ACK ");
   printf("\015\n Press 's' to continue");
   while ((((*HSTSTA_601B)&(0x01))!=1)&&(command!='s'))
   {
      if (getc()=='s')
      {
         command='s';
      }
   }
   /* Check HSTSTA Error Flags */
   fncterr=(*HSTSTA_601B)&(0x003E);
   if (fncterr!=0)
   {
      printf("\015\n Continuous Run Starting Error.");
      retval=-1;
   }
}
/* print 601B status registers for diagnostic purposes */
printf("\015\n HSTSTA: %x ",*HSTSTA_601B);
printf("\015\n ADSTAT: %x ",*ADSTAT_601B);

return(retval);
}

/**********************************************************
   startcom
      This function starts a command previously entered
into the 601B DPR.

**********************************************************/
startcom()
{

short int execerr;
```

```
/*  Reset Host acknowledge bit 0 & awaken the DVME 601 */
asm( "  CLR.W$FAFFF6" );
asm( "  MOVE.W#$0000,$FAFFFC " );

/*  Wait more than a few microseconds (about .1msec) */
asm( "  MOVEM.LD0-D7/A0-A6,-(A7) ");
asm( "  MOVEM.L(A7)+,D0-D7/A0-A6 ");
asm( "  MOVEM.LD0-D7/A0-A6,-(A7) ");
asm( "  MOVEM.L(A7)+,D0-D7/A0-A6 ");
asm( "  MOVEM.LD0-D7/A0-A6,-(A7) ");
asm( "  MOVEM.L(A7)+,D0-D7/A0-A6 ");
asm( "  MOVEM.LD0-D7/A0-A6,-(A7) ");
asm( "  MOVEM.L(A7)+,D0-D7/A0-A6 ");
printf("\015\n Delay For 601B IRQ Command ");


/* Verify the Executive is ON*/
execerr=0;
if (((*ADSTAT_601B)&(0x8000))!=0x8000)
{
   execerr=1;
   printf("\015\n EXEC ON not detected");
   printf("\015\n ADSTAT: %x",*ADSTAT_601B);
}
/* Check to see if requested operation acknowledged */
if (*HSTSTA_601B!=1)
{
   execerr=2;
   printf("\015\n HSTSTA Acknowledge Slow");
   printf("\015\n HSTSTA: %x",*HSTSTA_601B);
}

/* Return error code if error occurred,
   otherwise 0*/
if (execerr>0)
   return(execerr);
else
   return(0);

}

/***************************************************************
   dprsrecord
      This function writes the S-record file to be used by
local 601B memory to the low part of DPR memory.  The
function writes from lowest reserved S-record memory upward
($FA0000-$FAFFEF).

***************************************************************/
dprsrecord(command,ch)
char  command,ch;
{
```

```
/*  Increment the supplied address (supplied as a pointer)
    on each call of the function
    */
/* According to DATEL specifications there must only be one
LF or CR character between S-records, hence only the 0x0A
character is recognized.  */

if (ch!=0x0D)/* Skip any CR characters */
{
   if (command=='l')
   {
      *dpraddrs=ch;     /* Place an s-record in memory */
      dpraddrs=dpraddrs+1;
   }
   else
      if (command=='i')
         dpraddrs=DPRLO_601B; /* Initialize DPR pointer */
   }
}

/**********************************************************
   testsensor
      This function tests the A/D, D/A, and encoder boards
directly.  It can also be used to test the PUMA interface
board by comparing PIB output and input signals (measured
directly) with those read from the VMEbus A/D and those
written from the VMEbus D/A.

***********************************************************
testsensor()
{

int i,choice;
short int chanval;
float chanvolt;

/* NOTE: Only those A/D D/A channels used by this
controller are tested in this program  */

printf("\015\n ANY LOG Processes TERMINATED ");
logoff();
printf("\015\n******************************************");
printf("\015\n         SENSOR TESTING ROUTINE           ");
printf("\015\n");
printf("\015\nPress 'm' at any time to return to mainmenu");
printf("\015\n\015\n Select Test Mode");
printf("\015\n1.DVME601B A to D Channel Display 0..5");
printf("\015\n2.DVME601B A to D Channel Display 6..12");
printf("\015\n3.DVME628V D to A Channel Output 0..6");
printf("\015\n4.Encoder Board Channel Display");
prompt();
choice=' ';
```

```
while (choice!='m')
{
   choice=getc();
   switch(choice)
   {
          case '1': /* Display A to D Channels 0..5 */

     printf("\015\n A to D Test Channels 0..5 VOLTS");
               printf("\015\n Type 'm' to quit");
               printf("\015\n");
               while (choice!='m')
               {
                    i=0;
                    printf("\015");
                    while (i<6)
                    {
                        chanval=*(CHANNEL_601B+i);
                        chanvolt=chanval/3276.8;
                        printf(" %8.3f",chanvolt);
                        i=i+1;
                    }

                    choice=getc();
               }
               break;

          case '2':  /* Display A/D channels 6..12 */
               printf("\015\n A to D Test Channels 6..12");
               printf("\015\n Type 'm' to quit");
               printf("\015\n");
               while (choice!='m')
               {
                     i=6;
                     printf("\015");
                     while (i<12)
                     {
                        chanval=*(CHANNEL_601B+i);
                        chanvolt=chanval/3276.8;
                        printf(" %8.3f",chanvolt);
                        i=i+1;
                     }
                     /* Status Word    */
                     chanval=*(CHANNEL_601B+12);
                     printf(" %x",chanval);

                     choice=getc();
               }
               break;

          case '3':
             printf("\015\n D to A Test");
   printf("\015\n All Channel Voltages Changed Together");
```

```
             printf("\015\n Toggled 0..10");
             printf("\015\n 'm' to exit\015\n");
             while (choice!='m')
             {
                conbin[1]=0; conbin[2]=0; conbin[3]=0;
                conbin[4]=0; conbin[5]=0; conbin[6]=0;
                dacontrol();
                armon();

                printf("\0150V....");

                conbin[1]=0x7FF0; conbin[2]=0x7FF0;
                conbin[3]=0x7FF0; conbin[4]=0x7FF0;
                conbin[5]=0x7FF0; conbin[6]=0x7FF0;
                dacontrol();
                armoff();

                printf("\015....10V");
                choice=getc();
             }
             break;

          case '4':  /* Display all 6 encoder channels */
             printf("\015\n Encoder Test");
   printf("\015\n All Channels Monitored Simultaneously");
             printf("\015\n 'm' to exit\015\n");
             while (choice!='m')
             {
                rencoder();
          printf("\015%7d %7d %7d",enc[1],enc[2],enc[3]);
                printf(" %7d %7d %7d",enc[4],enc[5],enc[6]);
                choice=getc();
             }
             break;
   }
}
}

/*
   NOTES:
   (1) Prior to this routine the robot arm must be in a
position which is near its center so that it may move
unobstructed.

   (2) The robot arm must be turned on with a reqarmon()
command just prior to this routine being called.
*/

/************************************************************
  calibrate
    This function calibrates the encoder count to
  correspond to some absolute angle.  First the function
```

```
causes the joint 1..n to move until an index pulse is found
or until an encoder count greater than one index spacing has
passed.  It then takes the pot reading at this index pulse
and calculates what the encoder value should be at this
pulse.  The joint is then run back over this index and the
encoder is thus properly initialized.

**************************************************************/
calibrate()
{

int i,j,err,encerror,statval,foundit,potsum,jscan;

short int encval,encval2,potval,temp,temp2;

/* Mask to enable an index pulse, mask to see if one has
   arrived and the number of encoder pulses between index
   pulses for a given joint   */
short int indxmask,indxchk,indexres;

/* Number of encoder pulses from scribed calibrate marks to
   first index pulse*/
short int sdiffer;

/* Voltage between index pulses and voltage at the
   reference index pulse */
float potspan,indvref;


/* Loop through 6 joints */
/* Exit if an error condition exists or arm power off */
i=1;
err=0;
while ((i<7)&&(err==0)&&(conoff==0))
{
     printf("\015\n Cal. Joint: %d ",i);

     /* Obtain mask to activate proper index channel
     and other data about index spacing and pot to
     index spacing voltage ratio                     */
     switch(i)
     {
        case 1:
          indxmask=0xFFFE;
          indxchk=0x0001;
          indexres=1000;
          jscan=-2000;
          potspan=0.0435;
          indvref=2.5;
          sdiffer=673;
          break;
        case 2:
          indxmask=0xFFFD;
          indxchk=0x0002;
          indexres=800;
          jscan=-1800;
          potspan=0.0411;
          indvref=2.505;
          sdiffer=224;
          break;
        case 3:
          indxmask=0xFFFB;
          indxchk=0x0004;
          indexres=1000;
          jscan=-2000;
          potspan=0.0384;
          indvref=2.490;
          sdiffer=642;
          break;
        case 4:
          indxmask=0xFFF7;
          indxchk=0x0008;
          indexres=1000;
          jscan=-2000;
          potspan=0.0391;
          indvref=2.505;
          sdiffer=229;
          break;
        case 5:
          indxmask=0xFFEF;
          indxchk=0x0010;
          indexres=1000;
          jscan=-2000;
          potspan=0.0431;
          indvref=2.49;
          sdiffer=608;
          break;
        case 6:
          indxmask=0xFFDF;
          indxchk=0x0020;
          indexres=1000;
          jscan=-2000;
          potspan=0.0428;
          indvref=2.49;
          sdiffer=821;
          break;
      }

      /* Load index load register with the encoder value   */
      /* Make an est. of an approximately correct value   */
      /* Calc. the aver. value of the joint pot. voltage */
      j=1;
      potsum=0;
      while (j<9)
```

```
     {
        potsum=potsum+*(CHANNEL_601B+i-1);
        j=j+1;
     }
     potval=potsum*1.0/(j-1);

encval=encbypot(potval,potspan,indexres,indvref,sdiffer);

     /* Send encoder position estimate to encoder load reg.*/
     *INDLOAD_ENC=encval;

     printf(" INITIAL PULSE FIND ");

     /* Get current encoder value */
     *LATCH_ENC=0xFFFF;
     newsppos[i]=*(INDLOAD_ENC+i-1);
     sppos[i]=newsppos[i];


     /* TEMPORARILY CHANGE MAXIMUM CONTROL DEVIATION */
     /* DANGER DANGER DANGER !!!!!!!!!!!!!!!!!!!!!!!! */
     devmax=32000;

     /* Unmask index bit for selected joint*/
     *UNMASK_ENC=indxmask;

     /* Move past next index point to trigger index load */
     newsppos[i]=newsppos[i]+jscan;

     encerror=abs(sppos[i]-newsppos[i]);
     statval=((*STATUS_ENC)&(0x003F));
     foundit=0;

     /* Wait for index pulse or end of trajectory,
      whatever comes first */
     while ((encerror>50)&&(statval==indxchk)&&(conoff==0))
     {

        /* Check for joint move out of envelop  */
        if ((*(CHANNEL_601B+i-1)>potmax[i]) ||
          (*(CHANNEL_601B+i-1)<potmin[i]))
        {
          clkmask();
          conoff=1;   /* Stop robot arm, pot out of range */
          printf("\015\nROBOT POT DANGER ARM STOPPED!!! ");
        }
        else
        {
          encerror=abs(sppos[i]-newsppos[i]);
          /* Index arrival signified by bit set to 1 */
          statval=((*STATUS_ENC)&(0x003F));
          /* Has index pulse arrived? */
      if ((statval!=indxchk)&&(foundit==0))
      {
           /* YES! */
           foundit=1;
      }
    }
}
/* Make setpoint position equal to encoder count */
/* and stop.      */

/* Find present encoder count */
temp2=sppos[i];
*LATCH_ENC=0xFFFF;
temp=*(INDLOAD_ENC+i-1);
newsppos[i]=temp;
sppos[i]=temp;

if (foundit==1)
   printf("\015\n Calibration Pulse: %d Found ",i);

/* RESET MAXIMUM CONTROL DEVIATION */
/* END OF SAFETY HAZARD*/
/* ATTENTION ATTENTION ATTENTION !!!!!!!!!!!!!! */
devmax=DVMAX;

/* Check to see if index pulse found prior to above
     while loop */
if ((statval!=indxchk)&&(foundit==0))
{
printf("\015\n 1Fast Calibration Pulse: %d Found ",i);
   foundit=1;
}

printf(" # After first pulse%d #moved %d",temp,temp2);

/* Wait for joint to settle */
pause();

if ((foundit==0)||(conoff==1))
{
   printf("\015\n ERROR No Pulse Joint %d ",i);
   err=1;
}
else
{
  printf("\015\n Move 700 Points past");
  /* Move 700 points past this index point */
  newsppos[i]=newsppos[i]-700;

  /* Wait for joint to settle */
  while ((abs(newsppos[i]-sppos[i])>50)&&(conoff==0))
     { /* Do nothing */       }
```

```
        pause();

        printf("\015\n");

        /* Calc. the aver. value of the joint pot. voltage  */
        j=1;
        potsum=0;
        while (j<9)
        {
           potsum=potsum+*(CHANNEL_601B+i-1);
           j=j+1;
        }
        potval=potsum*1.0/(j-1);


        while  ((abs(newsppos[i]-sppos[i])>50)&&(conoff==0))
           { /* Do nothing (/)

        /* Load index load register with the encoder value
           for just passed index point  */

encval2=encbypot(potval,potspan,indexres,indvref,sdiffer);
        /* Adjust for displacement past index pulse */
        printf(" ENCVAL %d",encval2);

        /* Index load register initialized for next
           index pulse*/
        *INDLOAD_ENC=encval2;

        /* See what encoder count is prior next move */
        /* Diagnostic purposes  */
        *LATCH_ENC=0xFFFF;
        temp=*(INDLOAD_ENC+i-1);
        sppos[i]=temp;
        printf("\015\n Start Calib. encpos %d",temp);
        pause();

        /* Unmask index bit for selected joint*/
        *UNMASK_ENC=indxmask;

        /* Goto next index pulse*/
        newsppos[i]=newsppos[i]-indexres;

        statval=((*STATUS_ENC)&(0x003F));

        encerror=abs(sppos[i]-newsppos[i]);
        foundit=0;

        /* Wait for index pulse or end of trajectory,
           whatever comes first */
        while ((encerror>50)&&(statval==indxchk)&&(conoff==0))
        {
```

```
      encerror=abs(sppos[i]-newsppos[i]);
      /* Index arrival signified by bit set to 1 */
      statval=((*STATUS_ENC)&(0x003F));
      /* Index pulse found   */
      if ((statval!=indxchk)&&(foundit==0))
      {
           /* Yes ! */
         foundit=1;
      }
  }

  /* reset set points for new encoder position */
  temp2=sppos[i];
  *LATCH_ENC=0xFFFF;
  temp=*(INDLOAD_ENC+i-1);
  /* Stop at current set point */
  newsppos[i]=temp;
  sppos[i]=temp;
  if (foundit==1)
     printf("\015\n Calibration Confirmed JNT %d ",i);

  /* Diagnostic message */
  printf(" SPPOS STOP %d encpos %d",temp2,temp);

  if ((statval!=indxchk)&&(foundit==0))
  {
  printf("\015\n 2Fast Calibration Pulse: %d Found ",i);
     foundit=1;
  }

  if ((foundit==0)||(conoff==1))
  {
printf("\015\n Calibration of Joint %d Load Failure",i);
     /* Stop moving joint at current location */
     err=1;
  }
  else
  {
printf("\015\n Joint %d Calibration Successful ",i);
/* Move to zero of newly calibrated encoder count */
     newsppos[i]=0;

     *LATCH_ENC=0xFFFF;
     temp=*(INDLOAD_ENC+i-1);
     sppos[i]=temp;
     printf("\015\n temp to zero %d",temp);
     pause();

     /* Move to joint position zero encoder count */
     if (i!=2)
     {
   while ((abs(newsppos[i]-sppos[i])>20)&&(conoff==0)){}
```

```
          }
          else
          {
        while ((abs(newsppos[i]-sppos[i])>50)&&(conoff==0)){}
          }
          pause();
          printf(" index %d ",sppos[i]);

          }
     }
      i=i+1;   /* Calibrate next joint  */
}

/* Check to see if an error during calibration   */
if (err==1)
{
   printf("\015\n Abnormal Calibration Termination, ");
   printf("\015\n Results Invalid!");
}

*LATCH_ENC=0xFFFF;
sppos[i]=*(INDLOAD_ENC+i-1);
newsppos[i]=sppos[i];

pause();
pause();

/* Turn off system controller*/
clkmask(); /* Turn off periodic IRQ7 interrupts */
conoff=1;  /* Stop calls to control algorithm */
logoff();

/* Prompt user to redisplay main menu when ready */
printf("\015\n Type '1' to continue ");
prompt();
}

/***********************************************************
   pause
Wait for a while

***********************************************************/
pause()
{

int p;
p=0;
printf("\015\n PAUSE");
while (p<1000)
{
   printf("\015.");
   p=p+1;
```

```
}
}

/************************************************************
   encbypot
   This function returns the encoder count which corresponds
to the current pot value.

************************************************************/
encbypot(potval,potspan,indexres,indvref,sdiffer)
short int potval,indexres,sdiffer;
float potspan,indvref;
{

float potvoltage,potenc,indsign;
int indfind,encval;

/*What it the next index pulse load value expected? */
potvoltage=potval/3276.8;
printf(" POTVOLTAGE2 %f",potvoltage);

/* Calculate the encoder value for the
   next index pulse */
indsign=((indvref-potvoltage)/potspan);
indfind=indsign;
encval=indfind*indexres;
if (indsign<0)
   encval=encval-indexres;
potenc=encval+sdiffer;

return(potenc);
}
```

B.2.2 Skeleton.s Listing

```
;*************************************************************
; skeleton.s
;
; Skeleton Controller Functions:
; ==============================
;   These functions perform the low level operations required
;by the main 'c' language controller program.  When the
;controller is run the 'startup' code function stores the
;TUTOR registers, loads the new controller frame & stack
;pointers and branches to the main controller program which
;is written in 'c' language.  Functions exist to initialize
;the encoder, latch present encoder channel counts
;(rencoder), send voltage signals to the DtoA (dacontrol),
;start & stop the 68681 main system clock, allow/disallow the
;ARM power to be turned ON, and to initialize system
;hardware.  Other
;miscellaneous routines print A6 & A7 registers, print
;characters to the terminal, change AM line 3..5 values, jump
;back to TUTOR, and act as interrupt service routines.
;
;   IMPORTANT: DO NOT USE registers A6 or A7 to avoid
;conflicts with the 'c' program frame pointer or the stack
;pointer.
;
;*************************************************************
; External Reference:
;
.IMPORT _main; address of main 'c' routine
.IMPORTSTKTOP; start of user defined stack
;  comes from .spec file
.IMPORT FRAMETOP; user defined frame stack
;
; External Variables
;
;Encoder count variables:
;
.IMPORT _enc
;
;Encoder command variables:
;
.IMPORT _enc_command  ;= read, unmask, status, clear
.IMPORT _enc_mask      ;- unmasking encoder bits
.IMPORT _echannel; encoder channel being referenced
;
;Servo commands from controller
;
.IMPORT _conbin
;
;High level handler routine address
;
```

```
.IMPORT _handlerc
;
; Routines called by 'c' part of controller
;
.EXPORT _encoder,_rencoder,_dacontrol
.EXPORT _clkset,_clkunmask,_clkmask,_initial
.EXPORT _armoff,_armon
;
;*********************************************************
;
; System Device Address Table
;
;
;VMEbus AMline Controller
;
AMCODEEQU$ED0000 ; AMlines 3..5 latch address
;
;Encoder Board
;
CHAN1 EQU $020000 ; Encoder channel definition
CHAN2 EQU $020002
CHAN3 EQU $020004
CHAN4 EQU $020006
CHAN5 EQU $020008
CHAN6 EQU $02000A
STATUS EQU $02000C; Status register, bit=1
;  ==> waiting for index pulse
UNMASK EQU $02000E ; Set one bit of 6 low to allow index
; pulse to generate a load pulse
CLEAR EQU $020010 ; Clear the encoder counters, EPLDs
LATCH EQU $020012 ; Store the channel counter values
;
;TUTOR 68000, MC68681 DUART
;
BASE681 EQU $EFFC01
SERSTAT1 EQU $EFFC03 ; DUART for TUTOR 68000 terminal
SERTX1 EQU $EFFC07
SERRX1 EQU $EFFC07

TUTCPUS EQU $001000 ; Start of usable TUTOR CPU memory
;
;DMVE601B
;
;Note: All registers are word access
;
DPR601B EQU $FA0000 ; Start of 64K of DVME601B DPR
INT601B EQU DPR601B+$00FFFC ; VMEbus to 601B int. reg.
ID601 EQU DPR601B+$00FFFA ; Interrupt ID byte
; lower byte is ID
EINT601 EQU DPR606B+$00FFF8 ; Interrupt enable register
HSTSTA EQU DPR601B+$00FFF6 ; EXECutive status register
ADSTAT EQU DPR601B+$00FFF4 ;    "        "        "
```

```
AFBTOP EQU DPR601B+$00FFEF ; Appl. Function Block start
; AFB's build down in memory
;
;DVME628V       (Short Addressing)
;
DA_BASE EQU $FFC900 ; Base address
DA_OUT EQU $FFC9A0 ; Start addr. of 8 DAC output channels

;***********************************************************
;Data Sections
;
;***********************************************************
SECTION TUTORDATA
;
;Variables
;
SCRATCH DS.B 16; Working buffer
REGSAV DS.B 16; Reserved register saving area
;
;Start-up Routine storage
;
TSTACK DS.B 4; TUTOR STACK STORAGE
TFRAME DS.B 4; TUTOR FRAME STORAGE
;
;Data for startup code
;
     SECTIONram
_brkp DS.L 1
_brksz DS.L 1

;***********************************************************
;
; Main start up code subroutine:
;
; NOTE: This code was derived from an example startup code
; segment given in the 'c' cross-compiler manual.
;
;***********************************************************
     SECTION STARTUP

STARTC MOVE.L  A7,TSTACK  ; Store TUTOR stack
     MOVE.L    A6,TFRAME  ; Store TUTOR frame

;
;Addresses for stack & frame pointer come from .spec file
     LEA       STKTOP,A7   ;Initialize program stack
     LEA       FRAMETOP,A6 ;Init. program stack frame chain

;***********************************************************
;Zero out uninitialized RAM
************************************************************
     MOVE.L    #'BASE(ram),A1 ;point A1 to base of region ram
     MOVE.L    #'SIZE(ram),D0 ;set D0 to size of region ram
     LSR.L     #2,D0          ; compute size in longs
     BRA       ZDBF           ; enter a fast loop

ZLP: CLR.L     (A1)+; clear four bytes at a time
ZDBF:DBF       D0,ZLP; assumes 'SIZE(ram) < 256K bytes

;**********************************************************
; Initialize memory allocator
;**********************************************************
     MOVE.L    #'BASE(malloc),D0 ;address of malloc region
     MOVE.L    #'SIZE(malloc),D1 ;size of malloc region

   .IF"ptrd"?"2"
     SWAPD0; accommodate 2-byte C "pointers"
   .ENDIF
   .IF"long"?"2"
     SWAPD1; accommodate 2-byte C "longs"
   .ENDIF
     MOVE.L    D0,_brkp; variables referenced by mbrk()
     MOVE.L    D1,_brksz

;**********************************************************
;       Check stack & frame pointer for diagnostic purposes
;       Print values upon entry & exit from _main.
;**********************************************************
DIAGGO JSR     DIAGSF
;
;**********************************************************
;Invoke main() with no arguments.
;**********************************************************
     JSR       _main; any return value is "int" in D0
;
;**********************************************************
;       Check stack & frame pointer for diagnostic purposes
;**********************************************************
     JSR       DIAGSF
;
; Restore TUTOR stack and frame
;
     MOVE.L    TSTACK,A7
     MOVE.L    TFRAME,A6
     JSR       DIAGSF
     JMP       TUTOR
;
;**********************************************************
;
;  This section contains the library of functions called by
;the 'c' routine or those used at the assembly level
;
;**********************************************************
```

```
SECTION skeleton

;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;***********************************************************
; DRIVERS - routines directly interfacing with hardware
;
; NOTE: As it is expected that these routines are called by
;"c"  it is assumed that all registers are stored before
;these routines are called.  Also that any parameters placed
;on the stack by the calling routine are removed from there
;after the ;call.
;***********************************************************
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;
;***********************************************************
; ENCODER
;This  driver  accesses  all  the  encoder  functions  using
;character codes.  Parameters are passed to this routine by
;the global variables "mask" & "enc_command".  The mask is
;used to send an index arming command to the function when
;the "UNMASK" function is requested.  It is the number of the
;channel to be unmasked. The "STATUS" command can be issued.
;When the function returns, it is assigned a value of the
;index status register.If this function was not requested 00
;value is assigned.  When the READ function is selected all
;the counters are latched simultaneously and their value is
;stored in the 6 global position variables enc1..enc6. If the
;parameters  for  the  function  were  not  executable,  the
;function
;returns the value $FFFF.
;
; NOTE: The function value is returned in D0.
;
;***********************************************************
_encoder  MOVE.L #$39,D0
     JSR        AMCODER ;set up AMCODEs for standard addrs.

;
; Calling sequence according to "C" Compiler Conventions
;
     MOVE.B     _echannel,D2 ;get channel to be unmasked
     SUBQ.B     #1,D2
     ASL.B      #1,D2; convert channel to address index
     MOVE.B     _enc_command,D3  ; get commd char. string

     CMPI.B     #'r',D3
     BEQ        ENCREAD ; Read channels ?
     CMPI.B     #'u',D3
     BEQ        ENCMSK; Unmask a channel?
     CMPI.B     #'s',D3
     BEQ        ENCSTS; Status required ?
     CMPI.B     #'c',D3
     BEQ       ENCCLR; Clear encoder counters?
     MOVE.L #$FFFFFFFF,D0 ;Invalid parameters, flag error
     RTS
;
; 'r' Read the encoder channels, place in a global variable
;
ENCREAD MOVE.W #$FFFF,LATCH ; Save counter values of all
; channels
     MOVE.W    CHAN1,_enc+2   ; Array index 1..6
     MOVE.W    CHAN2,_enc+4
     MOVE.W    CHAN3,_enc+6
     MOVE.W    CHAN4,_enc+8
     MOVE.W    CHAN5,_enc+10
     MOVE.W    CHAN6,_enc+12
     BRA       ENCOUT0
;
; 'u'Unmask the selected channel
;
ENCMSK MOVE.W  #$FE,D3
     ROL.W     D2,D3
     MOVE.W    D3,UNMASK ;To unmask a channel set bit=0
     BRA       ENCOUT0
;
; 's'Provide the status register contents
;
ENCSTS MOVE.W  STATUS,D3 ;Send the status in return value
     SUB.W#1,D2
     BTST.L    D2,D3 ;Check to see if index pulse has
     BEQ  ENCINDX  ;arrived for selected channel
ENCNOIN MOVE.L #0,D0 ;No Index has yet arrived, function=0
     BRA       ENCEXIT
ENCINDX MOVE.L #1,D0 ;Index found, set function=1
     BRA       ENCEXIT
;
; 'c'Clear all encoder channel counters
;
ENCCLR MOVE.W  #$FFFF,CLEAR
ENCOUT0 MOVE.L #0,D0 ;No channel status requested
ENCEXIT RTS

;
;***********************************************************
;  rencoder
;     This function latches the encoder values & sends them
;to the global variables enc1..enc6 for use by the 'c'
;routine.  This function is quicker than the previous
;function because of lower overhead and is thus used for time
;critical situations.
;
;***********************************************************
_rencoder MOVE.W #$FFFF,LATCH
     MOVE.W    CHAN1,_enc+2
```

```
   MOVE.W   CHAN2,_enc+4
   MOVE.W   CHAN3,_enc+6
   MOVE.W   CHAN4,_enc+8
   MOVE.W   CHAN5,_enc+10
   MOVE.W   CHAN6,_enc+12
   RTS
;
;*****************************************************************
; dacontrol:
;       This routine sets DAC channels 1..6 to the 2's
;complement value specified in the conbin1..6 global
;variables. It is the user's responsibility to convert the
;control value to 2's complement. 2's complement is used by
;'c' int type variables and hence no conversion should be
;required in most cases. Channels are specified 1..6
;according to the PUMA joint specs.
;*****************************************************************
_dacontrol MOVE.L #$29,D0  ; Access short Non-priv. addrs.
   JSR      AMCODER
;
   LEAD     A_OUT,A0    ; Get D to A channel base address
   MOVE.W   _conbin+2,0(A0)  ; Send DAC channel voltage
   MOVE.W   _conbin+4,2(A0)  ; Array index 1..6
   MOVE.W   _conbin+6,4(A0)
   MOVE.W   _conbin+8,6(A0)
   MOVE.W   _conbin+10,8(A0)
   MOVE.W   _conbin+12,$A(A0)
;
   MOVE.L   #$39,D0  ;Restore AMcodes to default
   JSR      AMCODER  ;standard access type
   RTS
;
;*****************************************************************
; ARMOFF:
;Toggle OFF-L low to turn off robot arm
;
;*****************************************************************
_armoff
ARMOFF MOVEM.L D0/A0,-(SP)
   MOVE.L   #$29,D0; access DAC short addressing
   JSR      AMCODER

   LEA      DA_OUT,A0
   MOVE.W   #$7FF0,$C(A0) ;send +10V from to PIB OFF-HI
   MOVE.L   #$39,D0
   JSR      AMCODER; restore amline to standard access
   MOVEM.L(SP)+,D0/A0
   RTS
;
;*****************************************************************
; ARMON:
;Toggle OFF-L high to turn on robot arm
```

```
;
;************************************************************
 _armon
ARMON MOVEM.LD0/A0,-(SP)
    MOVE.L      #$29,D0 ;access DAC short addressing
    JSR         AMCODER

    LEA         DA_OUT,A0
    MOVE.W      #$0000,$C(A0); send 0V from to PIB OFF-HI
    MOVE.L      #$39,D0
    JSR         AMCODER; restore amline to standard access
    MOVEM.L(SP)+,D0/A0
    RTS
;
;************************************************************
;  AMCODER - set the bus AMCODES as requested.AM0..2 are
;controlled by the 680x0 CPU.  AM3..5 are set by this
;subroutine by writing to the AMCODE latch board.  The AMcode
;requested is stored in register D0.
;
; The AMcode generation board set the AM3,AM4,AM5 lines to
;a user defined value.  The user must write to the location
;AMCODE first  to set the latch signal low.  Next the user
;must write to the same location to set the AMCODE lines to
;the desired value.  Bit 0,1,2 correspond to the AM3, AM4,
;AM5 line levels.
;   Useful AMCODES: HEX CODEModifiersDescription
;
;   $3F,$3E,$3DH H H H x xStandard Super.
;   $3B,$3A,$39H H H L x xStandard No-priv
;   $2DH L H H L HShort Super.
;   $29H L H L L HShort No-priv
;
;************************************************************
AMCODER MOVE.L D0,-(SP)
    ROR.L       #3,D0; Access bits set by AMcode board
    ANDI.L      #7,D0; Isolate AM5,AM4,AM3 bits
    ORI.B       #%1000,D0; Set latch bit high
    MOVE.B      D0,AMCODE; Toggle latch bit high
    ANDI.B      #7,D0; Set latch bit low
    MOVE.B      D0,AMCODE; Latch AMcodes
    MOVE.L      (SP)+,D0
    RTS

;
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; LIBRARY ROUTINES:
;************************************************************
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;
;************************************************************
;  DIAGSF - perform diagnostics by printing out the stack
```

```
;pointer value and the frame pointer value.
;
;*************************************************************
DIAGSF    MOVE.L#8,D1
     MOVE.L    A7,D0
     ADDQ.L    #4,D0 ;Remove effect of the call to this subr.
     MOVEM.L   D0-D7/A0-A6,-(A7)
     JSR       HEXTOASC; print out stack pointer
     JSR       BUFPRNT
     JSR       LNEND
     MOVEM.L   (A7)+,D0-D7/A0-A6
     MOVE.L    A6,D0
     MOVEM.L   D0-D7/A0-A6,-(A7)
     JSR       HEXTOASC; print out frame pointer
     JSR       BUFPRNT
     JSR       LNEND
     MOVEM.L   (A7)+,D0-D7/A0-A6
     RTS
;
;*************************************************************
;  DEBUGP - this routine prints E0F to the terminal screen.
;This routine is to be used for debugging to trace the stream
;of execution. i.e. analogous to using 'printf' statements to
;see program execution flow.
;
;*************************************************************
DEBUGP MOVE.L  #3,D1
     MOVE.W    #$0E0F,D0
     JSR       HEXTOASC
     JSR       BUFPRNT
     JSR       LNEND
     RTS
;
;*************************************************************
;   TUTOR - return to the TUTOR prompt when TUTOR is the
;active monitor
;
;*************************************************************
TUTOR MOVE.B   #228,D7
     TRAP      #14
;
;*************************************************************
;  PUTCHAR - routine to write a character to the screen.
;WARNING: This routine will not return control to the
; calling program if the DUART locks up and the
; transmit register is never unbusy.
;
;*************************************************************
PUTCHAR BTST.B #02,SERSTAT1; Is Tx register busy?
     BEQ.S     PUTCHAR

     MOVE.B    D0,SERTX1; Send character to TX register
     RTS
;
;**********************************************************
;  HEXTOASC - this routine takes a hexidecimal number stored
;in D0 register and places the ASCII equivalent into a buffer
;pointed to by A6.  The least significant to most signifi-
;cant Hexidecimal digits are placed in the buffer in that
;order.The number of HEX digits is stored in D1.
;There are a maximum of 8 digits allowed.
;A6 points to one byte past the last byte entered in the
;buffer.  This routine adds a carriage return and line feed.
;**********************************************************
HEXTEST MOVE.L #$FEFEFEFE,D0
     LEA.L     SCRATCH,A6
     MOVE.L    #8,D1
     BSR       HEXTOASC ; Convert D0 to ASCII characters
     LEA.L     SCRATCH,A5
     BSR       BUFPRNT
     BRA       HEXTEST
;
HEXTOASC MOVE.L D2,REGSAV
     MOVE.L    D0,REGSAV+4
     MOVE.L    D1,REGSAV+8
     LEA.L     SCRATCH,A6
     MOVE.B    #$20,(A6)+
HLOOP SUB.L    #1,D1
     MOVE.L    D0,D2
     ANDI.L    #$0000000F,D2
     CMPI.B    #9,D2
     BHI       UPDIG
     ADD.B     #$30,D2
     BRA       BUFDUMP
UPDIG ADD.B    #$37,D2
BUFDUMP MOVE.B D2,(A6)+
     CMPI.B    #0,D1
     BEQ       HEXEND
     ROR.L     #4,D0 ;Get next HEX byte
     BRA       HLOOP
     MOVE.B    #$20,(A6)+ ;Add spaces to display
HEXEND MOVE.L  REGSAV,D2
     MOVE.L    REGSAV+4,D0; Restore register contents
     MOVE.L    REGSAV+8,D1
     RTS
;
;**********************************************************
;  BUFPRNT - This routine prints the contents of the buffer
;to the consol via the TUTOR 68000 board DUART.  The buffer
;start address must be in A6.  The buffer end address must be
;in A5.  The program moves from high address to low address.
;A6 points to one past the last byte stored.
;
;**********************************************************
```

```
BUFPRNT MOVE.L D0,REGSAV
     LEA.L     SCRATCH,A5
BUFBAK MOVE.B  -(A6),D0
BUFNTX BTST.B  #02,SERSTAT1    ; Test to see if TX busy
     BEQ.S     BUFNTX

     MOVE.B    D0,SERTX1; Send character to TX register
     CMPA.L    A5,A6
     BNE       BUFBAK; Continue emptying character buffer
     MOVE.L    REGSAV,D0
     RTS
;
;***********************************************************
;  LNEND - routine forces a carriage return and line feed
;    on the display terminal.
;
;***********************************************************
LNEND LEA.L    SCRATCH,A6
     MOVE.B    #$0D,(A6)+
     MOVE.B    #$0A,(A6)+; Load character buffer with CR, LF
     LEA.L     SCRATCH,A5
     BSR       BUFPRNT
     RTS
;
;***********************************************************
; initial.s:
;      This function initializes controller hardware.  It
;blocks the arm power from being turned ON until later in the
;'c' program when the control program is started.  It sets
;the AMlines on the VMEbus up for standard bus access mode.
;It clears the encoder board registers.  It sets the 68681
;clock up for cyclic interrupts to be used to clock the
;controller.
;
;***********************************************************
;
SECTION initial
_initial
;
     JSR       ARMOFF ;Do'nt allow ARM power to be turned on

     MOVE.L    #$39,D0
     JSR       AMCODER ;Set VMEbus for standard access
;
; Initialize Devices
;
     MOVE.W    #$FFFF,CLEAR ;Clear encoder board
;
     JSR       _clkset ;Timer setup for cyclic interrupts
     RTS
;
;***********************************************************
```

```
; _clkset
;This routine sets the MC68681 DUART to run its timer for a
;interval = 2 x (preload value) x (period of clock).
;
;The period of the external DUART clock is 3.6864 MHz.
;
;The periodic interrupts must be enabled by the user with the
;
; _clkunmask routine and disabled using the _clkmask routine.
;
;************************************************************
;
_clkset MOVE.L A4,-(SP)
     LEA       BASE681,A4
     MOVE.B    #%00110000,$8(A4) ;Put in count mode
     TST.B     $1E(A4) ;Stop counter command
     MOVE.B    #$07,$C(A4)    ;Load 1843 for 1msec period
     MOVE.B    #$33,$E(A4)    ; PRELOAD done  $0733,
                         ; b8=0.1msec
     MOVE.B    #31,$18(A4) ;Interrupt Vector = autovector 7
; Set interrupt mask 1=interrupt enable
     MOVE.B    #%00000000,$A(A4)
     MOVE.B    #%00000101,$4(A4)        ; Enable TXA,RXA
     MOVE.B    #%00000101,$14(A4)       ; Enable TXB,RXB
;
;  ACR: %01100000  div. by 1 timer enabled
;  ACR: %00110000  div. by 16 counter enabled
;   (disable the interrupts for this mode)
;
     MOVE.B    #%01100000,$8(A4) ;ACR:
     MOVE.L    (SP)+,A4
     RTS
;
;************************************************************
; _clkunmask
;This routine unmasks the MC68681 DUART timer driven
;interrupt so that it may be used for periodic control
;purposes.
;
;************************************************************
;
_clkunmask MOVE.L  A4,-(SP)
     LEA       BASE681,A4
     MOVE.B    #%00001000,$A(A4) ;1=timer interrupt enabled
     MOVE.L    (SP)+,A4
     RTS
;
;************************************************************
; _clkmask
;This routine masks all the MC68681 DUART timer interrupts
;so that active periodic tasks can be stopped while
parameters ;are being changed.
```

```
;
;**********************************************************
;
_clkmask MOVE.L A4,-(SP)
     JSR        ARMOFF ;For safety, turn robot arm off
     LEA        BASE681,A4
     MOVE.B     #%00000000,$A(A4) ;0=timer interpt.disabled
     MOVE.L     (SP)+,A4
     RTS

;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;**********************************************************
; EXCEPTION ROUTINES:
;**********************************************************
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;
;**********************************************************
; HANDLER
;   This interrupt service routine waits for IRQ7 to occur
;(68681 clock driven).  When the interrupt arrives it calls
;the 'c' handler routine and resets the timer IRQ status bit.
;
;**********************************************************
HANDLER MOVEM.L D0-D7/A0-A6,-(SP)
     LEA        BASE681,A4
     TST.B      $1E(A4) ;Reset Timer IRQ status bit STOPC

     JSR        _handlerc ;Call 'C' Handler

H1   MOVEM.L    (SP)+,D0-D7/A0-A6
     RTE

;
;**********************************************************
; BUSERROR (Bus error trapping routine)
;    Traps bus error & corrects the problem.  Not currently
;used. The TUTOR default routines are used.  This routine
;does work.
;
;**********************************************************
BUSERROR MOVE.L     A1,-(SP) ;save register A1 contents
     MOVE.L     SP,A1
     ADD.L      #6,A1 ;access current cycle address
     MOVE.L     (SP)+,A1 ;restore address register
;
; restore stack to past grandeur
;
     ADDQ   #8,SP
     RTE
;
;Unitialized exception routines
;
```

```
ADRERROR
ILLEGAL
PRIVILEGE
MISC
SPURIOUS
AUTO1
AUTO2
AUTO3
AUTO4
AUTO5
AUTO6
TRAP0
TRAP1
TRAP2
TRAP3
TRAP4
TRAP5
TRAP6
TRAP7
TRAP8
TRAP9
TRAPA
TRAPB
TRAPC
TRAPD
TRAPE
TRAPF
USER RTE
;**********************************************************
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
; EXCEPTIONS:
; ===========
;     Vector        Description         Classification
;     ======        ===========         ==============
;         2         Bus Error           Fatal
;         3         Address Error       Fatal
;         4         Illegal Instruction Fatal
;         5         Divide by 0         misc
;         6         CHK Instruction     misc
;         7         TRAPV Instruction   misc
;         8         Privilege Violation misc
;         9         Trace               misc
;         10        Emulation           misc
;         11        Emulation           misc
;         12-23     Reserved            misc
;         24        Spurious Interrupt  nonfatal
;         25        Level 1 Interrupt   service routine
;         26        Level 2 Interrupt   service routine
;         27        Level 3 Interrupt   service routine
;         28        Level 4 Interrupt   service routine
;         29        Level 5 Interrupt   service routine
;         30        Level 6 Interrupt   service routine
;          31        Level 7 Interrupt   service routine
;          32-47     Trap Instruction Vec. misc
;          48-63     Reserved            misc
;          64-255    User Interrupt Vectors  user
;
;***********************************************************
;
; Exception Vector Table:
;    Start at address $000060
;
;   NOTE: those vectors commented out use default TUTOR
;exception vectors. When porting to a different machine the
;user may have to write his or her own routines.
;
;***********************************************************
;
SECTION vects
;     DC.L BUSERROR        ;0x08
;     DC.L ADRERROR        ;0x0C
;     DC.L ILLEGAL         ;0x10
;     DC.L MISC,MISC,MISC  ;0x14
;     DC.L PRIVILEGE       ;0x20
;     DC.L MISC,MISC,MISC  ;0x24
;     DCB.L     12,MISC    ;0x30
      DC.L SPURIOUS        ;0x60
      DC.L AUTO1           ;0x64
      DC.L AUTO2           ;0x68
      DC.L AUTO3           ;0x6C
      DC.L AUTO4           ;0x70
      DC.L AUTO5           ;0x74
      DC.L AUTO6           ;0x78
      DC.L HANDLER         ;0x7C
;     DC.L TRAP0,TRAP1,TRAP2,TRAP3   ;0x80
;     DC.L TRAP4,TRAP5,TRAP6,TRAP7   ;0x90
;     DC.L TRAP8,TRAP9,TRAPA,TRAPB   ;0xA0
;     DC.L TRAPC,TRAPD,TRAPE,TRAPF   ;0xB0
;     DC.L MISC,MISC,MISC,MISC       ;0xC0
;     DC.L MISC,MISC,MISC,MISC       ;0xD0
;     DC.L 8,MISC                    ;0xE0
;     DC.L 192,USER                  ;0x100
```

B.2.3 User.c Listing

```
/**********************************************************
  user.c
     This 'c' program is used by the application programmer
to control the high level task execution of the robot.  It
calls external routines found in the robot controller kernel
to achieve its tasks. (e.g. movee, movea, moveta,
vectorje....)

**********************************************************/
extern int movee(int, int), movea(int, double), moveta(int,
double);
extern int vectorje(int,int,int,int,int,int);
extern int logon(void), logoff(void);
extern int getc(void), fputc(int,void *device),
clkmask(void);
extern int conoff;

user()
{

printf("\015\n Type 'g' to execute User Mode Trajectory ");
printf("\015\n");

while (getc()!='g')  { }
logon();

/* Move all joints simultaneously */
vectorje(4000,4000,4000,4000,4000,4000);
vectorje(-4000,-4000,-4000,-4000,-4000,-4000);

printf("\015\n Logging Completed");
logoff();

/* Move by 90 degrees then back again */
moveta(1,90.);  moveta(2,90.);  moveta(3,90.);

/* THESE MUST BE USED, NEGLECTING THEM MAY BE DANGEROUS
*/
clkmask();  /* Turn off periodic IRQ7 interrupts */
conoff=1;   /* Stop calls to control algorithms  */
}
```

B.2.4 Control.spec Listing

```
/************************************************************
Specification file for control.c & skeleton.s.

The entire program is RAM resident.

************************************************************/
partition { overlay {
    region {} vects;     /* other vectors */
} 01; } ROM[addr=0x0060];

partition { overlay {
    region {} STARTUP;   /* start up code   */
    region {} code;      /* executable code */
    region {} const;     /* constant data */
    region {} string;    /* constant strings */
    region {} skeleton;
    region {} initial;
    region {} TUTORDATA;
    DATA = $;            /* where to download region data */

    region {} data[roundsize=4]; /* RAM to be initialized*/
    region {} ram[roundsize=4]; /* RAM to be zeroed */
    region {} malloc[size=0x10]; /* RAM available to
                                    malloc() */
    region {} stack[size=0x1000]; /* stack of at least
                                        0x1000 bytes */
    STKTOP = $;          /* stack pointer reset value */
    region {} frame[size=0x1000];
    FRAMETOP = $;        /* frame pointer reset value */
} o1; } RAM1[addr=0x1000,size=0xE700];
```

B.2.5 Sunlink.c Listing

```
/*************************************************************
  sunlink.c
    This program provides a communication bridge between the
    TUTOR 68000 & SUN 3/160 systems via the RS-232 link
    connecting these systems.  This program is to run on
    the SUN system and is to be started remotely by the VME
    controller system which issues a UNIX command to run
    or to stop this program.

    This link has the following purposes:

    (1)Data logging - sending data to a user defined data
    file in the SUN system once it has been buffered in
    real-time by the TUTOR 68000 system.

    (2)601B local program "s" record - transferring of a
    68010 "s" record program file from the SUN 3/160 file
    system to the 601B "s" record DPR buffer memory using
    the TUTOR 68000 as the intermediary transmission port.
    This "s" record program will serve as the local 601B
    program.

    NOTES:

    (1) I/O Operating System Level Read/Write Commands:

             nchars = read(ft,buf,1);
             write(ft,buf,1);
    (2) The user can exit from this program at any time
        using the cntrl-c sequence.

*************************************************************/
#include <stdio.h>
#include <sgtty.h>
#include <sys/file.h>
#include <sys/time.h>
#include <errno.h>
#include <signal.h>

#define ETX 003
#define ENQ 005
#define ACK 006

struct sgttyb arg2;
struct tchars argtchars2;
qquit();
int ft;

char buf[1],nchars;
```

```
/*********************************************************
   main()
      This function sets up the SUN 3/160 communication port
B for the transfer of data in raw byte form without
buffering. It then calls the function sunlink() to wait for
data transfer instructions.

*********************************************************/
main()
{

struct sgttyb arg;
struct tchars argtchars;
char  c;
int nfds, width, exceptfds;
struct timeval timeout;
int i;
char hex[3];

/* Trap interrupt signals */
signal(SIGINT, qquit);


/* Open terminal and port A for reading and writing */
ft = open("/dev/tty",O_RDWR);
/* fd = open("sunlink.c",O_RDONLY); */

/* Get current settings of user terminal */
ioctl(ft,TIOCGETP,&arg2);
ioctl(ft,TIOCGETC,&argtchars2);

arg = arg2;

/* Set terminal characteristics */
arg = arg2;
arg.sg_flags |= CBREAK; /* CBREAK MODE */
arg.sg_flags &= ~ECHO; /* Turn off echo */

ioctl(ft,TIOCSETN,&arg);

/* Reset interrupt character */
argtchars = argtchars2;
argtchars.t_intrc ='^X';

ioctl(ft,TIOCSETC,&argtchars);

/* Terminal Emulation */
width = sizeof(int) * 8; /* no. of bits in masks */
timeout.tv_sec = 0;
timeout.tv_usec = 0;

/* Run SUN to VME controller communications package */
```

```
sunlink();
}
/***********************************************************
   qquit()
      This function restores terminal settings and exits

***********************************************************/
qquit()
{

ioctl(ft,TIOCSETN,&arg2);
ioctl(ft,TIOCSETC,&argtchars2);
exit();
}
/***********************************************************
   sunlink()
     This function provides a communication bridge between
the TUTOR 68000 & SUN 3/160 systems via the RS-232 link
connecting these systems.

***********************************************************/
sunlink()
{

int ch;

int bridgeoff;

/* Signal that the program is running okay     */
printf("\015\n  SUNLINK Bridge Program Executing ");
putchar(006);
bridgeoff=0;  /*Communication bridge active */

/* Poll to see what command the bridge is to execute */
while((ch=getchar())!= EOF)
{
    if (ch==ENQ)
    {

       /* Waiting for a command */
       bridgeoff=0;
       while(bridgeoff==0)
       {
          ch=getchar();
          switch(ch)
          {
               case 's':

                 /* VME system is sending a file to be
                    stored in the SUN file system    */
                 printf("\015\nSUN Send Option Executing");
                 receivefile();
                 bridgeoff=1;
                 break;

               case 't':

                 /* Send a file to the VME system which
                   was stored previously in the SUN file
                   system */
            printf("\015\nSUN Transmit Option Executing");
                 sendfile();
                 bridgeoff=1;
                 break;

               default:

                 /* Correct character has not arrived,
                      do nothing  */
                 break;
          }
       }
   }
   else { /* Do nothing */}

}
}

/**********************************************************
  receivefile:
    This program coordinates the transfer of files from the
VME controller to the SUN workstation.

This subroutine executes in the following sequence:
    (1) Write to VME controller (ACK)  to say okay to get
        filename .
    (2) Get the filename, stop getting once (ENQ) received
    (3) Open the file regardless of whether it already exists
    (4) Write to VME controller (ACK) to say file is open
        or (NAK) to say that there was a problem opening
        file.
    (5) Read characters until end of file.
    (6) When VME controller gets to the end of file it will
        send(ETX) to indicate the file transmission is
        complete. At this time the program should perform an
        fclose() operation on the open file.

    Since UNIX has difficulty receiving LF or CR characters
    the function waits for a '+' character to indicate an end
    of line.  When this character is received a LF is sent to
    the SUN logfile.

**********************************************************/
receivefile()
```

```
{

FILE *fd,*fopen();
int chrindex;
char ch,ch2,chrbuf[20];

/* Tell VME controller it is okay to send file name  */
printf("\006");     /* ACK - acknowledge character     */

/* Get filename until ENQ sent  */
chrindex=0;
/* Wait for first character to arrive */
ch=getchar();

while ((ch!=ENQ)&&(chrindex<=18))
{
   chrbuf[chrindex]=ch;
   chrindex=chrindex+1;
   ch=getchar();
}
/* Append string terminator so the array can be used as
   a string to open a file  */
chrbuf[chrindex]=0x00;
chrindex=chrindex+1;

if ((fd=fopen(&(chrbuf[0]),"w+"))==NULL)
{
    printf("\025");    /* Not okay file unopenable*/
}
else
{
    printf("\006");     /* Okay, file opened */
    ch=getchar();
    while (ch!=ETX)/* Check to see if end of file */
    {

      if (ch=='+')/* Check for special LF substitute */
      {
         ch=0x0A;
         fputc(ch,fd);
      }
      else
      {
         fputc(ch,fd);
      }
      ch=getchar();
    }
}

fclose(fd);
}
/************************************************************
```

```
  sendfile:
     This program coordinates the transfer of files from the
SUN workstation to the VME controller.

This subroutine executes in the following sequence:
    (1) Write to VME controller (ACK)  to say okay to get
        filename
    (2) Get the filename, stop getting once (ENQ) received
    (3) Open the file if it exists
    (4) Write to VME controller (ACK) to say file is open
        or (NAK) to say that there was a problem opening
        file.
    (5) Get a character from the file opened on the SUN.
        If the character is LF or CR change them to 0x0C and
        0x0B to avoid transmission problems.
    (6) Send a character to the VME controller, then wait for
        echo
    (7) If echoed character is the same as the sent character
        send(ACK), otherwise send ENQ character to see what
        is wrong.  Try this ten times before giving up.  Send
        NAK to indicate a serious transmission error then ETX
        to stop transmission.
    (8) Continue getting characters until end of file or
        retries exceed 10.
    (9) When SUN gets to the end of file, send
       (ETX) to indicate the file transmission is complete.
       At this time the program should perform an fclose()
       operation on the open file.

***********************************************************/
sendfile()
{

FILE *fd,*fopen();
int chrindex,retry;
char ch,ch2,chrbuf[20],buf[1];

/* Tell VME controller it is okay to send file name   */
printf("\006");    /* ACK - acknowledge character   */

/* Get filename until ENQ sent  */
chrindex=0;
ch=getchar();
while ((ch!=ENQ)&&(chrindex<=18))
{
    chrbuf[chrindex]=ch;
    chrindex=chrindex+1;
    ch=getchar();
}
/* Append string terminator so the array can be used
   as a string to open a file */
chrbuf[chrindex]=0x00;
chrindex=chrindex+1;

if ((fd=fopen(&(chrbuf[0]),"r"))==NULL)
    printf("\025");        /* Not okay file unopenable*/
else
{
    printf("\006");   /* Okay, file opened */
    retry=0;

        /* Continue sending characters until End of File */
    while (((ch=fgetc(fd))!=EOF)&&(retry<10))
    {
          if (ch==0x0A) /* Conv. to avoid 0x0D,0x0A seque.*/
             ch=0x0C;
          else
             if (ch==0x0D)
                ch=0x0B;

          buf[1]=ch;
          putchar(ch);

          /* Check char. echo to see if good transmission */
          ch2=getchar();
          if (ch2==ch)
             printf("\006");  /* ACK, last character okay */
          else
          {
             /* Prepare for retransmission     */
             retry=1;
             while((ch2!=ch)&&(retry<10))
             {
               printf("\005");  /* ENQ sent   */
               printf("%c",ch); /* Resend lost character */
               ch2=getchar();  /* Wait for reply */
               if (ch==ch2)
                  printf("\006");  /* Ack. ok transmit */
               else
                  retry=retry+1;
             }
             if (retry>=10)    /* Error occurs  */
             printf("\025");    /* End of transmission */
          }
    }
}

printf("\003");/* ETX - signal that all file read */

fclose(fd);
}
```

B.2.6 Program601b.s Listing

```
;
;  program601b.s
;      This program runs in local 601B memory once it has
;  been downloaded from the DPR S-record buffer.
;  It performs the following functions:
;
;(1) Takes a four point average of POT and Control line
;    signals to eliminate false steady state (low band-
;    width signals).  This reduces the chance of false
;    triggering of Control line error conditions.
;(2) It copies the current measurement signal directly
;    to DPR without any preprocessing.
;(3) It sends an IRQ6 to the main system processor if
;    an error condition in control lines is detected.
;
     SECTION        LOCAL601
LOCAL     JMP       GOCHK
;
; Data Section
;
AVERCNT DC.W   0    ;count number of samples in average
AVERBUF DC.L   [16]0 ; running sum 4 byte x up to 16 samples
; initialize all sums to zero
;
; Tests for PUMA power amp & brake status
;
SSRONBAD  EQU  $1999 ; > 2 volts logic 1, error condition
STOPLBAD  EQU  $1999 ; > 2 volts arm off
BRKRLSBAD EQU  $0CCD ; < 1 volt indicates brake is ON

BUFSTART  EQU  $00021000 ;Start of A/D buffer
DPRBUF    EQU  $00040000 ;Start of DPR RAM
;
; Starting Point
;
GOCHK

     MOVEM.L   D0-D3/A0-A3,-(A7) ;Save registers
     LEA.L     AVERBUF,A0
     LEA.L     BUFSTART,A1
     LEA.L     DPRBUF,A2
     CMPI.B    #03,AVERCNT ;has the 4th sample arrived?
     BNE       SUMMORE
AVERAGE CLR.L  D1   ;channels 0..11 monitored
     CLR.L     D2
     CLR.B     AVERCNT  ;initialize count for next summation
     CLR.L     D3
AVLOOP MOVE.W  (A1),D2 ;get latests A/D value for channel
     CMPI.B    #7,D1 ;Is this a current meas. sample?
     BNE       AVCONT
     MOVE.W    D2,(A2) ;send current meas. directly to DPR
     CLR.L     (A0) ;perform no averaging on current
     BRA       AVNEXT
AVCONT ADD.L   (A0),D2 ;perform last add of summation
     CLR.L     (A0) ;zero average memory for next sum
     LSR.L     #2,D2 ;compute average div. by 4
     MOVE.W    D2,(A2) ;send result to DPR buffer
LEVELCHK CMPI.B #8,D1  ;check for control line sample
     BLS       AVNEXT ;check for line status
     CMPI.B    #9,D1
     BNE       LEVEL2
     CMPI.W    #SSRONBAD,D2 ;If ssronhi* line high, system
     BLE  LEVEL2    ;is having a problem
     ORI.W     #$01,D3 ;brake, temperature or other fault
     BRA       AVNEXT
LEVEL2 CMPI.B  #10,D1
     BNE       LEVEL3
     CMPI.W    #STOPLBAD,D2 ;If signal high, arm power is
     BLE       LEVEL3 ;OFF, non-fatal condition
     ORI.W     #$02,D3
     BRA       AVNEXT
LEVEL3 CMPI.B  #11,D1
     BNE       LEVEL4
     CMPI.W    #BRKRLSBAD,D2 ;brake released , fatal!
     BGE       LEVEL4 ;if signal low, brake is released
     ORI.W     #$04,D3
LEVEL4 CMPI.B  #0,D3
     BEQ       AVNEXT ;see if any error conditions flagged
     MOVE.W    D3,DPRBUF+24 ;send status word to DPR location

AVNEXT ADDQ.L  #4,A0 ;increment buffer pointers
     ADDQ.L    #2,A1
     ADDQ.L    #2,A2
     ADDQ.B    #1,D1 ;increment to next channel
     CMPI.B    #12,D1 ;process only 12 channels
     BNE       AVLOOP
     BRA       DONECHK
;
;  Perform intermediate summation prior to signal averaging
;
SUMMORE CLR.L  D1
     CLR.L     D2
SUMLOOP MOVE.W (A1),D2
     ADD.L     D2,(A0) ; sum next element
     CMPI.B    #7,D1
     BNE       SUMNEXT
     MOVE.W    D2,(A2) ; send current meas. to DPR
;  directly, use no averaging
SUMNEXT ADDQ.L #4,A0
     ADDQ.L    #2,A1
     ADDQ.L    #2,A2
     ADDQ.B    #1,D1
     CMPI.L     #12,D1
     BNE        SUMLOOP ; has 12th channel been scanned?
     ADDI.B     #1,AVERCNT  ; increment the summation counter
DONECHK MOVEM.L (A7)+,D0-D3/A0-A3

     RTS                ;return to AFB loop

;
; This section starts the 68681 timer and lets it run
; continuously
; It is called by the initialization AFB.
;
     SECTION    CLKSTART
     MOVE.B     #0,$120019 ;Disable 68681 timer
     MOVE.B     #$7B,$12001F ;Set timer for 1msec interval
     MOVE.B     #02,$120019 ;div. by 10, delay mode start
     RTS
```